NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

DESIGN AND ANALYSIS OF AN ACCESS CONTROL
SYSTEM FOR A MULTI-BACKEND DATABASE SYSTEM

by

Ali Ekici

June 1984

Thesis Advisor: David K. Hsiao

Approved for public release; distribution unlimited

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Design and Analysis of an Access Control System for a Multi-backend Database System		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1984 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ali Ekici		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE June 1984 13. NUMBER OF PAGES 113
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Access control, database, attribute, backend		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis describes the design and analysis of an access control mechanism for a multi-backend database system (MDBS). MDBS utilizes a minicomputer as the controller and a number of minicomputers and their disk systems as the backends. The database is distributed over the dedicated disk systems of the backends. The operations on the database are performed by the backends in parallel. Thus, the performance gain of the system is dependent on the number of backends in the system. Each backend (Cont)

ABSTRACT (Continued)

performs its own access control operations using duplicated access control information.

Detail Design and Analysis of
an Access Control System
for a
Multi-backend Database System

by

Ali Ekici
Ltjg, Turkish Navy
B.S., Turkish Naval Academy, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June, 1984

ABSTRACT

This thesis describes the design and analysis of an access control mechanism for a multi-backend database system (MDBS). MDBS utilizes a minicomputer as the controller and a number of minicomputers and their disk systems as the backends. The database is distributed over the dedicated disk systems of the backends. The operations on the database are performed by the backends in parallel. Thus, the performance gain of the system is dependent on the number of backends in the system. Each backend performs its own access control operations using duplicated access control information.

TABLE OF CONTENTS

I. INTRODUCTION

The main contribution of this thesis is the design and analysis of an access control mechanism for the Multi-Backend Database System (MDBS). The design outlines of the access control mechanism have been suggested in [Ref.2]. In this thesis. the detailed design and analysis are given. To this end, we first present the general concepts of computer security. We then present the role of database security.

A. COMPUTER SECURITY

"Computer security deals with the managerial procedures and technological safeguards applied to computer hardware, software, and data to assure against accidental or deliberate unauthorized access to and dissemination of computer system data" [Ref.1]. The importance of computer security can best be appreciated if the computer is used to store and process the information about military secrecy or proprietary industrial items.

Computer security can be considered in four basic layers: physical security, hardware security, software security and database security. Physical security deals with the protection of computer against physical accesses. Physical access can be controlled with managerial, identification and authentication procedures. Hardware, software and database security are necessary only after physical access to the computer is accomplished. Hardware security deals with the protection of the user's data and program from other users by way of hardware protection mechanisms such as memory protection. Software security deals with the protection of the user's data and program by way of software protection mechanisms. The basic mechanisms for software security are surveillance, threat monitoring and access control. Surveillance is to keep track of the user and the associated resources which are requested by the user. Access control deals with the user access authorization for resources during the program execution. These resources can be programs, files, etc. Logically,the operating system handles the access control with an access control matrix indexed by user-ids and resource-ids where each entry of the matrix consists of authorizations for the user to access the corresponding resource.

Database security is concerned with the protection of databases stored in the computer. If data of a database must be kept confidential, then the database must be protected against unauthorized accesses. When a large amount of confidential data must be processed and protected, the issues of database security become very important.

Since our main subject is to provide an access control mechanism for a database system, we will elaborate the issues of database security in the following section. Other security issues will not be the subject for discussion in this thesis.

B. DATABASE SECURITY

The principal problem for database security can be defined as determining <u>who</u> can access <u>which</u> data and <u>what</u> are the operations allowed on the data. This may be thought of as the same as the basic resource-access control problem which has been presented above. In essence, the problem is handled with the similar techniques, but, here, the argument is the comprehension of the semantics of the data rather than the basic resource-ids. For example, the basic question for access control is to decide 'who' can perform 'what' operations on 'which' data in the database. First two questions can be argued for both the resource-access control and the database access control, but to specify 'which' portion of the the database is to be authorized for the users may be more involved with the data model and semantics of the data.

Database management systems treat data differently than traditional data processing systems. Traditional data processing systems store data as a collection of values but database management systems store data as a collection of attribute-value pairs and relationships among the pairs. The attributes allow us to view the types or characteristics of corresponding values. The relationships allow us to relate one type of values to another type of values.

In the database system under discussion (MDBS), we will consider database security for databases which are organized as attribute-value pairs. This gives us uniform representation of information and we can keep track of some semantic relationships among data.

C. ACCESS DECISION PROBLEMS OF DATABASE SECURITY

Access decision may be based on the following factors. Access decision on *value-sensitive* information depends on the current value of data and the user's authorized value limits. Access decision on *state-sensitive* information depends on the dynamic state of the database management system. For example, the user can open a file only if it is not in an unlocked state. There is *pattern-sensitive* information, such that, the user may be allowed to sort a file but he may not be allowed to read the content of the file. *History-sensitive* information must be protected against a series of operations which may allow some inferences about unauthorized information. For example, if ranks and salaries of persons in the database are coincident, then the user can read the rank of a person and infer the salary. *Event-sensitive* information must be protected from the user during the unauthorized period. For example, tellers can only access the system during the working hours.

The access control mechanism for a database management system should protect all types of information presented above. However, since protections of some of these types are difficult to implement, most systems do not implement for all the types.

D. ACCESS CONTROL PRECISION

After an access decision, the database management system must perform physical access to the database in order to fetch authorized and requested data. The database management system should access only authorized and requested data. This is necessary for performance and security reasons. A system which fetches only authorized and requested data is called a system with <u>absolute precision</u>. Absolute precision must be the ultimate goal of database management system designers.

Database management systems without absolute precision cause the <u>pass-through problem</u>, since they fetch some unnecessary data in order to access the authorized and requested data. If the data which have been passed through have more stringent security feature then there may be a security breach. The system should have either absolute precision or solve the bad aspect of the pass-through problem.

In the next two subsections, some basic approaches for the pass-through problem will be described. Detailed information can be found in [Ref.1].

1. Compartmentalization

The data with identical protection requirements are grouped and stored together. These groups with uniform security requirements are called security atoms. Security atoms become the unit for access. If we do not need all records in a security atom then we will have access imprecision. However, we will not have the pass-through problem, since none of the unnecessary records in the security atom have the higher level protection requirements than the authorized ones.

2. Multilevel Security

It may be desirable to classify the data in the database by hierarchical layers. The user may have access authorization up to a certain security level. This notion can be combined with the security atom concept. Each security atom may belong to a classification level. So, the combination of security atoms with the same security classification can be used to create these layers.

E. ACCESS CONTROL SYSTEM DESIGN METHODOLOGY FOR DATABASE SYSTEMS

We have seen that database security deals with the semantics of data. Before designing the access control system, the designer must study and analyze the system's data model. Since, most of the time, the data abstractions are very large and complex, this analysis must be very orderly and in only necessary detail. The designer has to look for a standpoint to start. It can be an uniform data unit for security atom concept, etc.

There is only one interface between the user and the database system, the request. The access control system must recognize requests and it must have a mechanism to start the access decision with information available in request.

In the next chapter, we will give an overview of MDBS in terms of data abstraction, data manipulation language and basic request execution process in order to understand the system and utilize the system facilities in the access control mechanism. In the third chapter, we will describe the basic design issues of MDBS access control system and the access control information requirements.In the following chapters, we will elaborate the the basic steps of the access control system.

II. AN OVERVIEW OF THE MULTI-BACKEND DATABASE SYSTEM (MDBS)

In this chapter, we will introduce the general structure of the multi-backend database system (MDBS). MDBS has been designed in [Ref.2] and [Ref.3], and the implementation of the system has been presented in [Ref.4], [Ref.5] and [Ref.6]. The information in this chapter has been extracted from [Ref.4] and [Ref.5].

A. A TOP-LEVEL VIEW OF MDBS

One minicomputer functions as the controller, with multiple minicomputers and their disks configured in a parallel manner to serve as backends [Ref.2, Ref.4]. The database is distributed on secondary storage across all of the backends. User access is accomplished through a host computer communicating with the controller. The MDBS structure can be classified as a centralized system, although the hardware is distributed.

As shown in Figure 2.1, the controller and the backends are connected by a broadcast bus. When a transaction is received from the host computer, the controller broadcasts the transaction to all the backends at the same time. Each backend has a number of dedicated disk drives. Since the data is distributed across the backends, a transaction can be executed by all backends in parallel. Each backend maintains a queue of transactions. When one transaction has been executed, the backend can begin execution on another transaction from its queue.

B. A TOP-LEVEL VIEW OF THE MDBS DATA MODEL

The data model chosen for the system is the attribute-based data model [Ref.2]. In MDBS the database consists of files of records. Each *record* is a collection of keywords, optionally followed by a record body. A *keyword* is made of an attribute-value pair such as <SALARY,$12,000> where $12,000 is the value of the attribute SALARY. A *record body* is a string of characters not used by MDBS for search purposes. An example of a record without a record body is shown below.

(<FILE,Employee>, <EMPLOYEE_NAME,Smith>, <CITY,Columbus>,
<SALARY,$12,000>, <SERVICE,10>)

The first attribute-value pair in all records of a *file* are the same. In particular, the attribute is FILE and the value is the *file name*. For example, the above record is from the Employee file.

1. *Keyword Predicates*

A *keyword predicate*, or simply *predicate*, is of the form (attribute, relational operator, value). A *relational operator* can be one of the set { =, !=, >, >=, <, =< }. A keyword K is said to *satisfy* a predicate P if the attribute

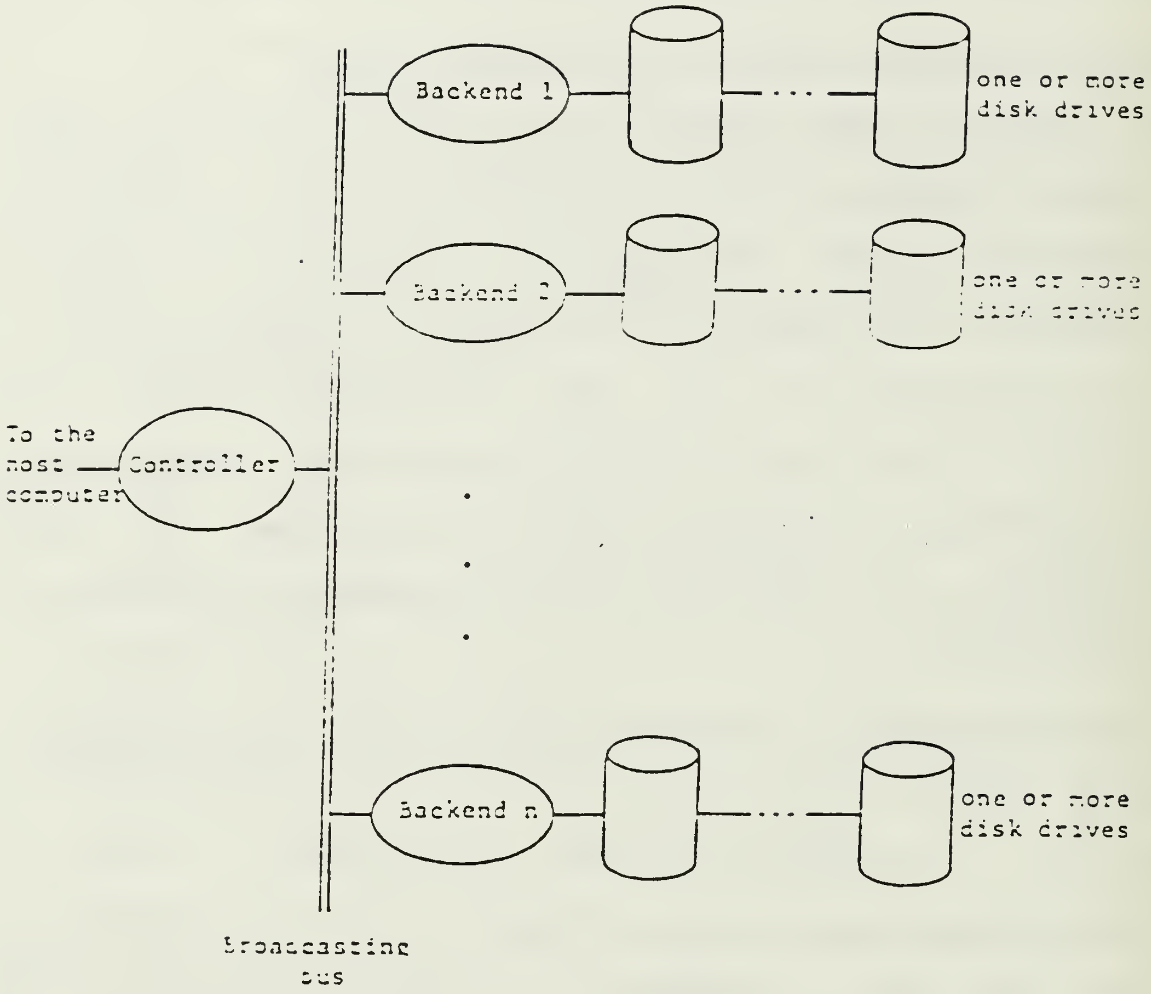


Figure 2.1 The MDBS Hardware Organization

of K is identical to the attribute in P and the relation specified by the relational operator of P holds between the value of K and the value in P. For example, the keyword <SALARY,15000> satisfies the predicate (SALARY > 10000).

2. Three Types of Descriptors

A descriptor can be one of three types: Type-A, Type-B and Type-C. A Type-A descriptor is a conjunction of a less-than-or-equal-to predicate and a greater-than-or-equal-to predicate, such that the same attribute appears in both predicates. An example of a type-A descriptor is as follows:

((SALARY >= 2,000) and (SALARY =< 10,000)).

More simply, this is written as (2,000 =< SALARY =< 10,000).

A Type-B descriptor is an equality predicate. An example of a type-B descriptor would be (POSITION = Manager).

A Type-C descriptor consists of only an attribute name, known as the type-C attribute. Let us assume that there are n different keywords K1, K2, ..., Kn, in the records of a database with a type-C attribute. Then, this type-C descriptor is really equivalent to n type-B descriptors B1, B2, ..., Bn, where Bi is the equality predicate satisfied by Ki. In fact, this type-C descriptor will cause n different type-B descriptors to be formed. From now on, we shall refer to the type-B descriptors formed from a type-C descriptor as type-C sub-descriptors. For instance, consider that DEPT is specified as a type-C attribute for a file of employee records. Furthermore, let all employees in the file belong to either the Toy department or the Sales department. Then, two type-C sub-descriptors for DEPT will be formed as follows:

(DEPT = Toy) and (DEPT = Sales)

3. The Relationship of Keywords and Descriptors

A keyword is said to be derived or derivable from a descriptor if one of the following holds:

(1) The attribute of the keyword is specified in a type-A descriptor and the value is within the range of the descriptor.

(2) The attribute and value of the keyword match those specified in a type-B descriptor.

(3) The attribute of the keyword is specified in a type-C descriptor.

4. The Cluster Formation

For performance reasons, records are logically grouped into clusters based on the attribute values and attribute value ranges in the records. As described above, these values and value ranges are called descriptors. For example, one cluster might contain records for those employed in Columbus, making at least $20,001 but not more than $25,000 and with at least 11 but not more than 15 years of service. Thus records of this cluster are grouped by the following three descriptors:

(CITY=Columbus),($20,001=<SALARY=<$25,000),(11=<SERVICE=<15)

These descriptors are said to define the cluster which contains records of employees in Columbus making between $21,000 and $22,000 per year and with 12 to 13 years experience would require the retrieval of records in the cluster just described. Other requests, such as to find records of employees in Columbus making between $21,000 and $28,000 and with 12 to 13 years experience, might require additional retrieval of records from other clusters than the one identified above.

In MDBS, processing is done a cluster at a time. In order to allow efficient processing of requests, records in a cluster are spread across all the backends. Thus each backend needs to search only its portion of the cluster. Given a user request, there must be a way, of course, first to determine which clusters to search and then to determine the location of records in a given cluster. To perform this task, MDBS utilizes available descriptor information. For example, given the previous request for finding employees where

```
    (CITY=Columbus)and($21,000=<SALARY=<$28,000)
                                  and(12=<SERVICE=<13)
```

MDBS first determines that two clusters must be searched. These are the clusters identified by the two sets of descriptors:

```
    {(CITY=Columbus),($20,001=<SALARY=<$25,000),
                                    (11=<SERVICE=<15)}


    {(CITY=Columbus),($25,001=<SALARY=<$30,000),
                                    (11=<SERVICE=<15)}
```

After the clusters are identified, MDBS must then determine the disk addresses of the clusters at each backend. Finally MDBS will cause each backend to retrieve from its disks the records so addressed.

<u>Descriptor search</u> determines the descriptors that correspond to the request. In our example, there are four descriptors corresponding to the request; namely,

(CITY=Columbus), ($20,001=<SALARY=<$25,000),

($25,001=<SALARY=<$30,000), (11=<SERVICE=<15).

In order to save space and to save processing time each descriptor is identified by a descriptor id. For example,

Descriptor	Descriptor Id
(CITY = Columbus)	D15
(\$20,001 =< SALARY =< \$25,000)	D125
(\$25,001 =< SALARY =< \$30,000)	D126
(11 =< SERVICE =< 15)	D250

Thus the output of the descriptor search phase is the Boolean expression of descriptor ids

$$\text{D15 and (D125 or D126) and D250} \qquad (1)$$

corresponding to

$$\text{(CITY=Col) and } \begin{array}{c}\text{(\$20,001=<SALARY=<\$25K)} \\ \text{or} \\ \text{(\$25,001=<SALARY=<\$30K)}\end{array} \text{ and (11=<SERVICE=<15)}$$

which identifies two clusters. The next phase, cluster search must take the Boolean expression in (1) and actually determine the corresponding clusters. As with descriptors, clusters are also identified by ids, known as cluster ids, for example

Descriptor Ids	Cluster Id
D15, D125, D250	C17
D15, D126, D250	C22

The final two phases are address generation (to find the disk addresses, e.g., A3546 and A3547, corresponding to each cluster id, e.g., C17) and record selection (to retrieve the actual records so addressed).

Descriptor search, cluster search and address generation together form the major portion of directory management.

Because all directory management is based on the concept of clusters, it is also logical to design an access control capability based on clusters. Thus cluster search is augmented by a cluster access control mechanism.

C. THE DATA MANIPULATION LANGUAGE

The data manipulation language for MDBS is a non-procedural language which supports four different types of requests - retrieve, insert, delete and update. The syntax of these various requests and examples of them are presented below.

1. Retrieve Requests

A retrieve request

```
RETRIEVE Query Target-List [BY Attribute] [WITH Pointer]
```

consists of five parts. The first part is the type of the request i.e. RETRIEVE. The second part is a query which identifies the portion of the database to be retrieved. A query is any arbitrary Boolean expression of predicates. An example of a query is:

```
((DEPT=Toy)and(SALARY<10000)) or
                    ((DEPT=Book) and (SALARY>50000))
```

The target-list is a list of elements. Each element is either an attribute, e.g., SALARY, or an aggregate operator to be performed on an attribute, e.g., AVG(SALARY). MDBS supports five aggregate operators - AVG, SUM, COUNT, MAX, MIN. The values of an attribute in the target-list are retrieved from all records identified by the query. If no aggregate operator is specified on the attribute in the target-list, its values in all the records identified by the query are returned directly to the user or user program. If an aggregate operator is specified on the attribute in the target-list, some computation is to be performed on all the attribute values in the records identified by the query and a single aggregate value is returned to the user or user program. The fourth part of the request, referred to as the BY-clause, is optional as designated by the square brackets around it. The use of the By-clause is explained by means of an example. Assume that employee records are to be divided into groups on the basis of the departments for the purpose of calculating the average salary for all the employees in a department. This may be achieved by using a retrieve request with the specific target-list, (AVG(SALARY)), and the specific BY-clause, BY DEPT. Finally, the fifth part of the request, which is an optional WITH-clause, specifies whether pointers to the retrieved records must be returned to the user or user program for later use in an update request. Some examples of retrieve requests are presented below.

Example-1

Retrieve the names and salaries of all employees making more than $1000/month.

```
RETRIEVE (FILE=Employee) and (SALARY>5000) (NAME,SALARY)
```

Example-2

List the average salary of all departments.

```
RETRIEVE (FILE=Employee) (AVG(SALARY)) BY DEPT
```

2. <u>Insert Requests</u>

An insert request

```
INSERT Record
```

specifies a record to be inserted into the database. An example of an insert request is:

```
INSERT (<FILE,Employee>,<SALARY,5000>,<DEPT,Toy>)
```

3. <u>Delete Requests</u>

A delete request is of the form:

```
DELETE Query
```

where the Query specifies the particular records to be deleted from the database. An example of a DELETE request is:

```
DELETE (NAME=Smith) or (SALARY>50000)
```

4. <u>Update Requests</u>

An update request is of the form:

```
UPDATE Query Modifier
```

where the Query specifies the particular records to be updated from the database and the Modifier specifies the kinds of modification that need to be done on records that satisfy the query. In an update request, if a single attribute value is to be changed, then the attribute is termed the <u>attribute being modified</u>. The modifier in an update request specifies the new value to be taken by the attribute being modified. The new value to be taken by the attribute being modified is specified as a function f of the old value of either the same attribute or some other attribute (say, attribute-1). More specifically, the modifier may be one of the following five types:

Type-0 : <attribute=constant>

Type-I : <attribute=f(attribute)>

Type-II : <attribute=f(attribute-1)>

Type-III : <attribute=f(attribute-1) of Query>

Type-IV : <attribute=f(attribute-1) of Pointer>

D. THE MDBS HARDWARE AND SOFTWARE ORGANIZATION

MDBS is implemented in several permanent processes. The process structure within the controller and each backend is shown in Figure 2.2.

In the following two sections, we describe the functions performed in the controller and the backends, respectively.

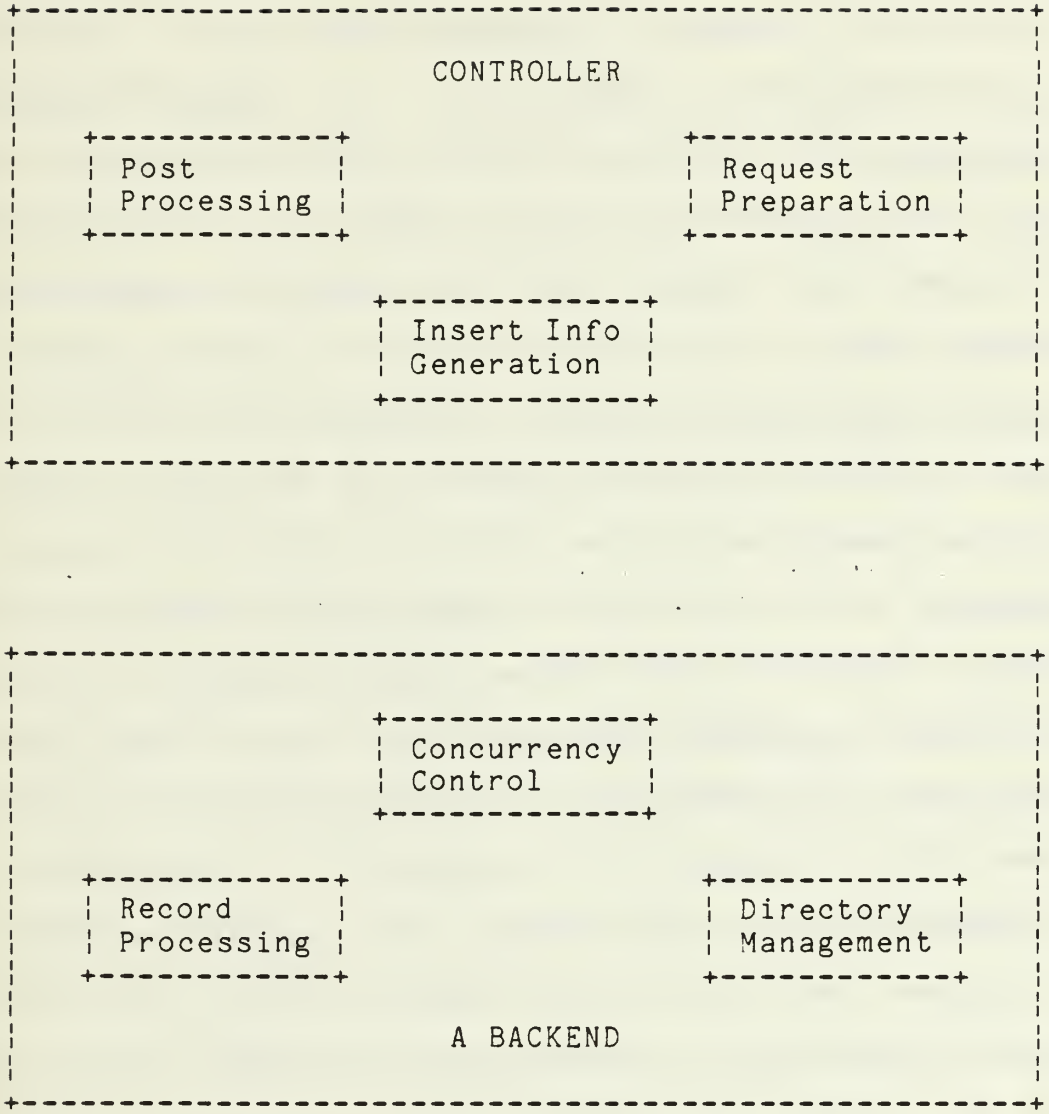


Figure 2.2 - The Process Structure in the Controller and the Backends.

1. Functions of the Controller

The MDBS controller consists of three categories of functions: request preparation, insert information generation and post processing. The request preparation functions are those which must be performed before a request or a transaction can be broadcasted to the backends. For example, each request must be parsed and checked for syntax errors before it can be broadcasted to the backends. The insert information generation functions are those which must be performed during the processing of an insert request to furnish additional information required by the backends. For example, a backend should be selected for storing the record being inserted into the secondary storage of the backend. The post processing functions are those which must be performed after replies are returned from the backends, but before the results of a request or a transaction are forwarded to the host machine. For example, the results for a request returned by each backend should be collected. After receiving the results from each backend, the response to the request can be sent to the host machine.

We note that there are no concurrency control functions in the controller. Since user requests are carried out by the backends, there is no need for concurrency control in the controller. The controller must only associate sequence numbers with the user requests.

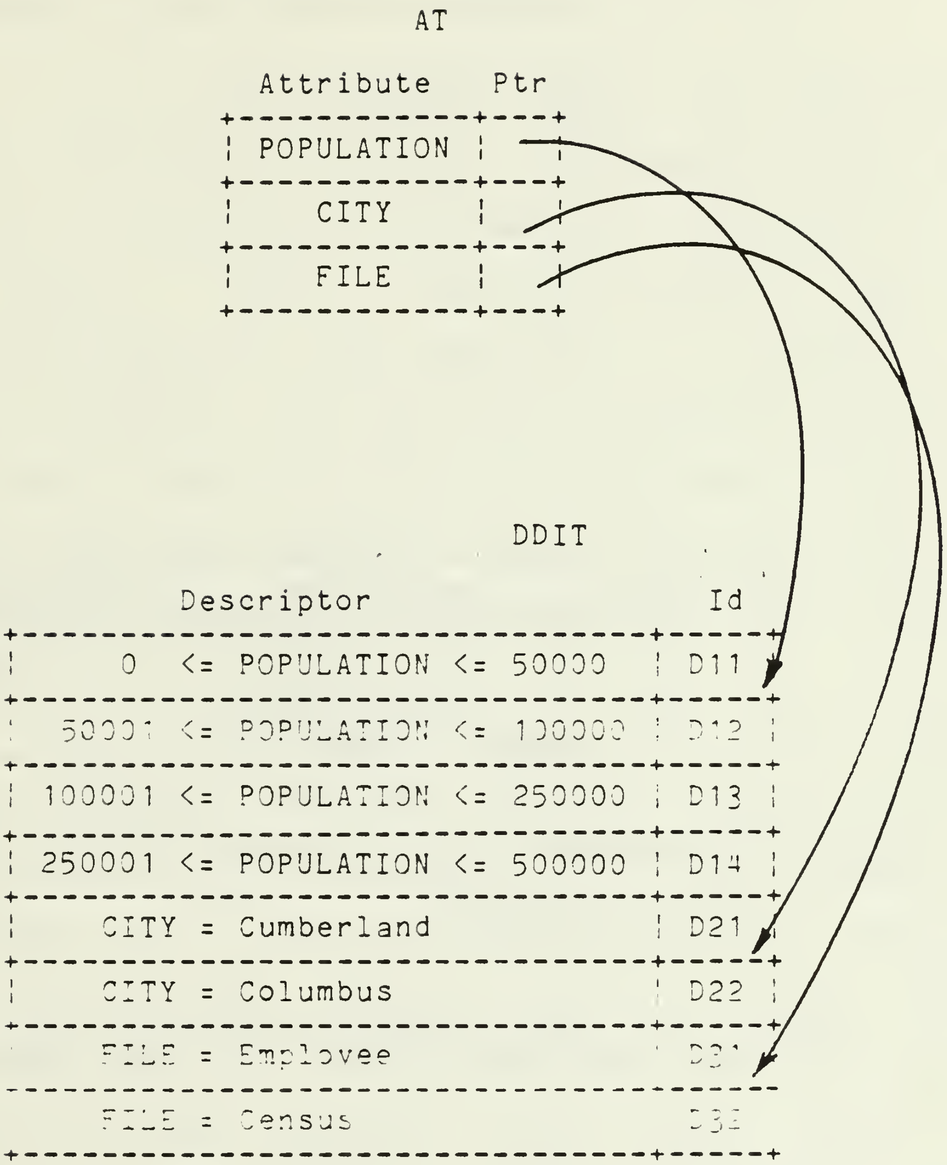
2. Functions of each Backend

Each backend in MDBS consists of three categories of functions: directory management, record processing and concurrency control. The directory management function performs descriptor search, cluster search, address generation and directory table maintenance. The record processing function performs record storage, record retrieval, record selection and attribute value extraction of the retrieved records. For example, these functions store records into the secondary storage, retrieve records from the secondary storage and select the records that satisfy a query from a set of records. The concurrency control function performs operations which ensure that the concurrent and interleaved execution of user requests will keep the database consistent. For example, these functions schedule a user request for execution based on the set of clusters needed by the request. Since, in this thesis, we deal with database security, handled in the backends, we will next describe the functions of the backends in more detail.

a. The Directory Management Function

Directory management has four phases: Descriptor Search, Cluster Search, Access Control and Address Generation. The input to directory management is either the record part of an insert request or the query part of a retrieve, delete, or update request. The three non-insert request types, namely, retrieve, delete and update, require the same directory management processing. However, the insert request type requires a different directory management processing. Thus we will describe directory management in terms of two categories: non-inserts and inserts.

Directory management has four directory tables: The descriptor-to-descriptor-id table (DDIT), the attribute table (AT), the cluster-definition table (CDT) and the cluster-Id-to-next-backend table (CINBT) These tables are an integrated part of the directory management. Logically, they are defined as follows: All the descriptors defined by the database creator are stored in the descriptor-to-descriptor-id table (DDIT). There is a descriptor id associated with each descriptor. There is an entry in the attribute table (AT) for every directory attribute. A pointer to the DDIT is stored with each directory attribute. The pointer points to the first descriptor whose attribute is identical to the corresponding directory attribute. A sample AT and DDIT are depicted in Figure 2.3. By showing these two tables together, we can easily depict the pointers of AT. The cluster-definition table has an entry for every cluster. Each entry consists of the cluster number, the set of descriptor ids whose descriptors define the cluster, and addresses of the records in the cluster (Figure 2.4). The



AT

Attribute	Ptr
POPULATION	
CITY	
FILE	

DDIT

Descriptor	Id
0 <= POPULATION <= 50000	D11
50001 <= POPULATION <= 100000	D12
100001 <= POPULATION <= 250000	D13
250001 <= POPULATION <= 500000	D14
CITY = Cumberland	D21
CITY = Columbus	D22
FILE = Employee	D31
FILE = Census	D32

Dij = Descriptor j for attribute i.

Figure 2.3 A Sample Attribute Table (AT) and Descriptor-to-Descriptor-Id Table (DDIT)

Cluster	Desc-Id Set	Address (*)
C1	{D11,D21,D31}	A1,A2
C2	{D14,D22,D32}	A3

(*) Secondary storage addresses for the records in the cluster

Figure 2.4 A Sample Cluster-Definition Table (CDT)

cluster-id-to-next-backend table (CINBT) is used as a basis of record insertion to the backends. Let us look at the four phases of the directory management.

(1) The Descriptor Search. In this phase, directory management determines the descriptor ids of the descriptors that satisfy the predicates (keywords) in a query (record). Input to Descriptor Search comes from Request Preparation in the controller. As described in detail in [Ref.5], if there are N backends processing a query (record) with X predicates (keywords), then each backend performs descriptor search on X/N predicates (keywords) and broadcasts the descriptor ids to the other backends.

(2) The Cluster Search. In this phase, directory management determines either the cluster id of the cluster to which a record belongs (for an insert request) or the cluster ids of the clusters whose records may satisfy a query (for a non-insert request). Input to Cluster Search are the descriptor ids found by Descriptor Search in all the backends. For insert requests, Cluster Search passes the cluster id, if any, to Insert Information Generation in the controller. For non-insert requests, the cluster ids are passed to Address Generation.

(3) The Address Generation. In this phase, directory management determines either the secondary storage address for storing a record when processing an insert request or the addresses of all the records in a set of clusters when processing a non-insert request. For insert requests, Insert Information Generation in the controller determines which backend is to insert the record. When a backend is selected, Address Generation in that backend determines the secondary storage address for record insertion. That address and the formatted request are then passed to Physical Data Operation.

For non-insert requests, Cluster Search passes the cluster ids to Address Generation. Address Generation finds the addresses of the records in these clusters and passes the addreses and the formatted request to Physical Data Operation.

b. The Record Processing Function

This function perform operations such as record selection and field extraction of the retrieved records. The names of these operations are: Physical Data Operation and Aggregate Operation.

(1) The Physical Data Operation. Input to this operation comes from Address Generation. The input is a set of secondary storage addresses and the request. Physical Data Operation performs different actions depending on the type of the request. For an insert request, Physical Data

Operation stores the record being inserted into the secondary storage.

For a non-insert request, i.e., delete, retrieve or update, Physical Data Operation fetches the records from the secondary storage and selects the records that satisfy the query in the request. It then performs the intended operation on the basis of the type of the non-insert request. For delete requests, Physical Data Operation marks the selected records for deletion.

For retrieve requests, Physical Data Operation extracts the values from the selected records. If an aggregation is to be applied to the request, then Physical data Operation passes the values to Aggregate Operation phase of Record Processing.

For update requests, Physical Data Operation updates the selected records and returns to the secondary storage those updated records that have not changed cluster. If one or more records change cluster, Physical Data Operation marks the old records for deletion and sends the records that have changed cluster to Request Preparation in the controller. Request Preparation initiates the actions required for the insertion of these records into their new clusters.

(2) The Aggregate Operation. This operation performs the partial aggregate operations in retrieve requests. Input to Aggregate Operation comes from Physical Data Operation in the form of a set of values and the aggregate operators to be applied. Aggregate Operation applies the aggregate operations on the set of values and passes the results to Post Processing in the controller.

In chapter I, we introduced the access control systems in general. In chapter II, we gave an overview of the structure and the attribute-based data model of MDBS. In the next chapter, we will describe the access control mechanism of MDBS.

III. ACCESS CONTROL MECHANISM OF MDBS

We have seen that MDBS has four major phases for directory management: descriptor search, cluster search, access control and address generation. In descriptor search, MDBS determines the descriptor-id groups which identify the clusters requested by the user. In cluster search, MDBS determines the cluster-ids corresponding to the descriptor-id groups found at descriptor search. The access control phase eliminates the unauthorized cluster-ids. Finally, address generation determines secondary storage addresses for the authorized clusters. In this chapter, we describe the access control phase in more detail.

A. THE ACCESS CONTROL PRINCIPLES OF MDBS

The access control mechanism [Ref.2], has been designed to utilize the parallel execution facilities of the backends. Let us see the basic principles of MDBS's access control mechanism.

1. Access Control must Utilize the Advantages of the MDBS Architecture

The basic idea of the MDBS architecture is to have a multiple backend system which performance most of the database management functions in the backends. Because of this architectural principle, the access control mechanism is added to the function of the backends. Thus, we must store the security-related information in the backends. Each backend only needs to store the subset of tables, which are pertinent to data stored in that backend. As a result, if the response time and the throughput of the backend system will be improved by parallel backend processing, then the access control function will also be improved accordingly.

2. Access Control System should not Decrease Efficiency

An access control mechanism amy bring additional overhead to a system. It might be thought that additional access control checks on the records would slow down the request execution. However, We do not agree with this observation. If we have a mechanism to distinguish the authorized records from the unauthorized ones before record processing, then we need not fetch the unauthorized records. Consequently, we do not require secondary storage accesses for unnecessary, unauthorized records. In this way, we can improve the efficiency of the system for some requests. In addition, this mechanism provides absolute precision and avoids the pass-through problem.

3. Access Control System should not Provide only Static Security Policiec for Dynamic Databases

The database creator should have an ability to specify the protection policy for each user at database creation time. In addition, the database creator should also have the ability to specify the protection policy for new users. Since a database will not be static due to insertion, deletion and update of records, we want to have an access control mechanism to create new security specifications which fit the database creator's security policy.

4. Access Control should Provide Uniform Protection for Record Collections of Same Characteristics

We want to protect records with the same characteristics in the same way. Records with similar access control requirements should be stored together and isolated from other records with different access control requirements. Recall that clusters are the collections of records with uniform characteristics. We want to store similar records together in a cluster. We can extend the concept of clusters to handle security. If we have security specifications on a cluster then we can protect all records in this cluster in the same way. Such a grouping of records has been called a security atom in chapter I. Utilization of the security atom concept will solve the pass-through problem also.

5. Access Control System should Provide Finer Granularity

In request execution with access control, MDBS should also be restrict an operation on particular records in a cluster. For example, the access control system should prevent the update operations, if a record in a cluster has a salary which is greater or equal to $2000. This feature brings access control to the field level (i.e., the level of attibute-value pairs) from the cluster or record level. With the field level access control, the mechanism provides a finer granularity of security.

In MDBS, values of an attribute may be divided into different ranges. Each such attribute-range pair is called a descriptor. If we use descriptors for security specifications, then we can protect the domain of the attribute values of the database. Since clusters are defined by sets of mutually exclusive descriptors on each attribute, we can also protect the clusters by employing their defining descriptors. Descriptors are created at the database creation time. Thus, the security specifications can also be created at the database time. If we want to have finer granularity on a particular attribute for access control, we can create descriptors with smaller attribute-value ranges.

B. THE ACCESS CONTROL REQUIREMENTS OF MDBS

MDBS should have an access control mechanism which protects value-dependent and pattern-sensitive information. History-sensitive and event-sensitive information is not considered as subject of the MDBS access control system. State-sensitive information is dealt with concurrency control mechanism.

Value-dependent access control protects data on the basis of the relationship of the attribute value of the data and the user of the data. It is not enough to have protection on the value of a descriptor as the finest granularity of security. For example, the database creator may want to allow managers to retrieve the salary of employees in their respective departments. If we authorize the managers to retrieve the salary of employees, then the managers will have authorization to retrieve the salary of all employees whether or not they are in the managers' department. Thus, the access control must be dependent on specific attribute value in order to provide value-dependent security.

Another requirement for MDBS access control system is to provide security for statistics of the database by controlling the execution of requests which utilize the aggregate functions such as average, maximum and minimum.

C. THE ACCESS CONTROL STEPS

The access control checks described above occur at two disjoint times. The access control checks which are performed before any record retrieval are considered as pre-processing while those which are performed after the record retrieval are considered as post-processing. As has been stated in the second access control principle, pre-processing is one of the goals of the access control system.

In the next section, we discuss pre- and post-processing in more detail.

1. Pre-processing

Access control checks made prior to the retrieval of records from the secondary storage prevent the access control mechanism from having to deal with unauthorized records. Thus, access control precision is increased. To achieve this goal, we have to utilize information about the records available in the directory management, namely, the clusters information. Thus, we should perform some access control checks before the record processing phase and even before the address generation phase.

In the preprocessing, the access-control checks are related to descriptors and clusters. We can utilize the descriptor and cluster information to perform pre-processing by having some security specifications specified by the database creator on the descriptors.

2. Post-processing

We recall that the records in a database consist of attribute-value pairs. Some of these attributes are chosen as directory attributes. Descriptors are then defined on these attributes in order to build clusters. The other attributes are non-directory attributes which are not used for defining clusters. The database creator may also want to specify some security specifications on these attributes. In addition, an attribute may become more important because of the dynamic nature of the database. The database creator may want to specify some security specifications on this attribute but may not want to redesign the database by making the attribute into a directory attribute.

The non-directory attribute values of the records are not reflected in their directories. Consequently, the only way to determine them is to examine the attribute values of every retrieved record. This process can be performed after the record retrieval as a post-processing operation.

Allowing security specifications on the non-directory attributes degrades the precision of the access control. If the database has no security specifications on the non-directory attributes, then the access control system can achieve absolute precision. Consequently, the database creator must consider this situation. If certain attribute values should be protected, then their attributes should be defined as directory attributes. Nevertheless, we still want to provide a mechanism for post-checking in order to give flexibility to the database creator for later use of non-directory attributes for access control of the database. The MDBS post-processing mechanism would not affect absolute precision severely, since there is no reason to use non-directory attributes for security specifications at the database creation time. It is built in for allowing flexibility in use by the database creator.

D. THE PREPROCESSING ACCESS CONTROL INFORMATION AS SPECIFIED BY THE DATABASE CREATOR

In this section, we describe the preprocessing access control information which is specified by the database cretor at the database creation time. We also describe how this information is stored in MDBS.

In order to decide on the authorization for a request, we need the information concerning who can perform what operations on which data in the database. The database creator can specify the first two types of information easily. For example, for specifying 'who', the database creator can utilize user-ids which can be provided by MDBS. For specifying 'what', the database creator can select some of the access operations from the MDBS data manipulation language. Specifying 'which' portion of the database is a more elaborate exercise requiring a more detailed knowledge of the data model. In the rest of the section, we will show to specify a 'portion' of the database in order to give some access rights on it.

We know that a cluster is the basic unit of access in MDBS. As was argued in [Ref.2], we also want to utilize the cluster as the basic unit for access control. Thus, we expect the database creator to make some security specifications at the cluster level. However, the database creator is not directly awared of the cluster formation. The database creator only specifies directory attributes and disjoint descriptors on these attributes. MDBS then forms the clusters for the database creator. Consequently, we can expect the database creator to specify the security specifications in terms of descriptors.

We must find a way to transform the access control descriptors (about the fields) to the authorized and unauthorized cluster-level (about the records) This transformation will be addressed again later in this chapter. First we will discuss the specification and storage of access control descriptors.

1. Access Control Descriptors

The database creator specifies field-level access controls on the descriptors for each user. A field-level access control specification is a triple of the form:

(Aggregate operator, Attribute, Disallowed access operation)

The form of a field-level access control specification has been slightly changed because of some implementation purposes, but the basic semantics is the same as given in [Ref.2].

The attribute in the field-level access control specification may be a directory or a non-directory attribute. The dissallowed access operation is one of the set {No_Retrieve, No_Delete, No_Insert, No_update}. The aggregate operator is one of the aggregate operators of the MDBS data manipulation language (e.g., Max, Avg, Sum etc.).

The aggregate operator may be omitted in the specification. The attribute is not necessary for a field-level access control when the disallowed access operation is No_Insert or No_Delete, since we insert and delete the whole record not only an attribute of the record.

A security specification (ss) for a descriptor may be 'all' or 'null' or a collection of field-level access control specifications. A ss of 'all' indicates that all accesses are disallowed for the respective user. A ss of 'null' indicates that no accesses are disallowed for the respective user. The ss on a descriptor D is the collection of field-level access control specifications for the descriptor D. It can be represented as follows:

```
(D) {<Aggr_op, Attr_id, dissallowed access op>,<---->
     ......<---->}
```

For example, we might have the following descriptor-ss for the descriptor D:(0<Salary<100) is

```
(D)  { <--,Job,No_Update>, <Avg,Salary,No_Retrieve>,
       <--,Name,No_Retrieve>, <--,--,No_Delete>,
       <--,--,No_Insert> }
```

The meaning of this security specification is as follows: if a record has a salary between zero and 100, then the user can not update the job attribute of the record. Nor can the user retrieve the average value of salaries and Name of the employees of all the records whose salaries are in the range of zero and 100. In addition, the deletion and insertion of any record with salary between 0 and 100 is not authorized.

We store all the descriptor-based security specifications for each user in a descriptor-to-security-specification table (DSST). This table and its implementation are discussed in the next two subsections

2. The Descriptor-to-Security-Specification Table (DSST)

Logically, DSST can be incorporated into an augmented DDIT by adding to DDIT one column per user. See Figure 3.1 below. Instead, we would like to store the descriptor-level access conrol information in a seperate descriptor-to-security-specification table as shown in Figure 3.2.

There are several reasons to have a seperate table. First, the descriptor-based security specifications are database-creator-specified access control information and are used to create cluster-level access control information. They are not used directly for request authorization. We have decided that MDBS would use cluster-level access control information for request authorization. Such cluster-level access control information is created once, at the database creation time. It is used whenever we need to

Descr id	Descriptor	SS for user-1	SS for user-2
D1	0<Salary=<100	{<Avg,Salary,No_Retrieve>}	none
D2	0>Salary>100	{<---,Name, No_update>}	all
...		....	...

Figure 3.1 An augmented DDIT

Des-id	SS for user_1	SS for user_2
D1	{<Avg,Salary,No_retrieve>}	none
D2	{<---,Name, No_update>}	all
...	....	...

Figure 3.2 A separate Descriptor-to-Security-Specification Table (DSST)

deal with that cluster. The descriptor-level access control information is then no longer used directly.

The second reason for a seperate DSST is that these security specifications are of variable length whereas the DDIT entries are of fixed length. Consequently, it is more appropriate to separate variable length entries from fixed length length entries, if we do not use them at the same time in an operation.

The third reason for a separate DSST is that each user may have a distinct ss on a descriptor. All these distinct security specifications must be stored. If the database has a large number of users, then the main function of DDIT which is 'the mapping from descriptor to descriptor-id' will be very inefficient.

3. The Secondary-Storage-Based DSST

Each user in the database will have a column in DSST as depicted in Figure 3.2. If the database has a large number of users, then we may want to store the DSSTs in the secondary storage. Except as discussed in Chapter V, we use the DSST only at the database load time to create the cluster-based security specifications. Thus, it will be efficient to store the DSST in the backends' secondary storage. We can just fetch the particular user's DSST or part of the DSST when it is needed.

Figure 3.3 shows a sample of the secondary-storage based DSST. Each database in MDBS will have a _user-index_

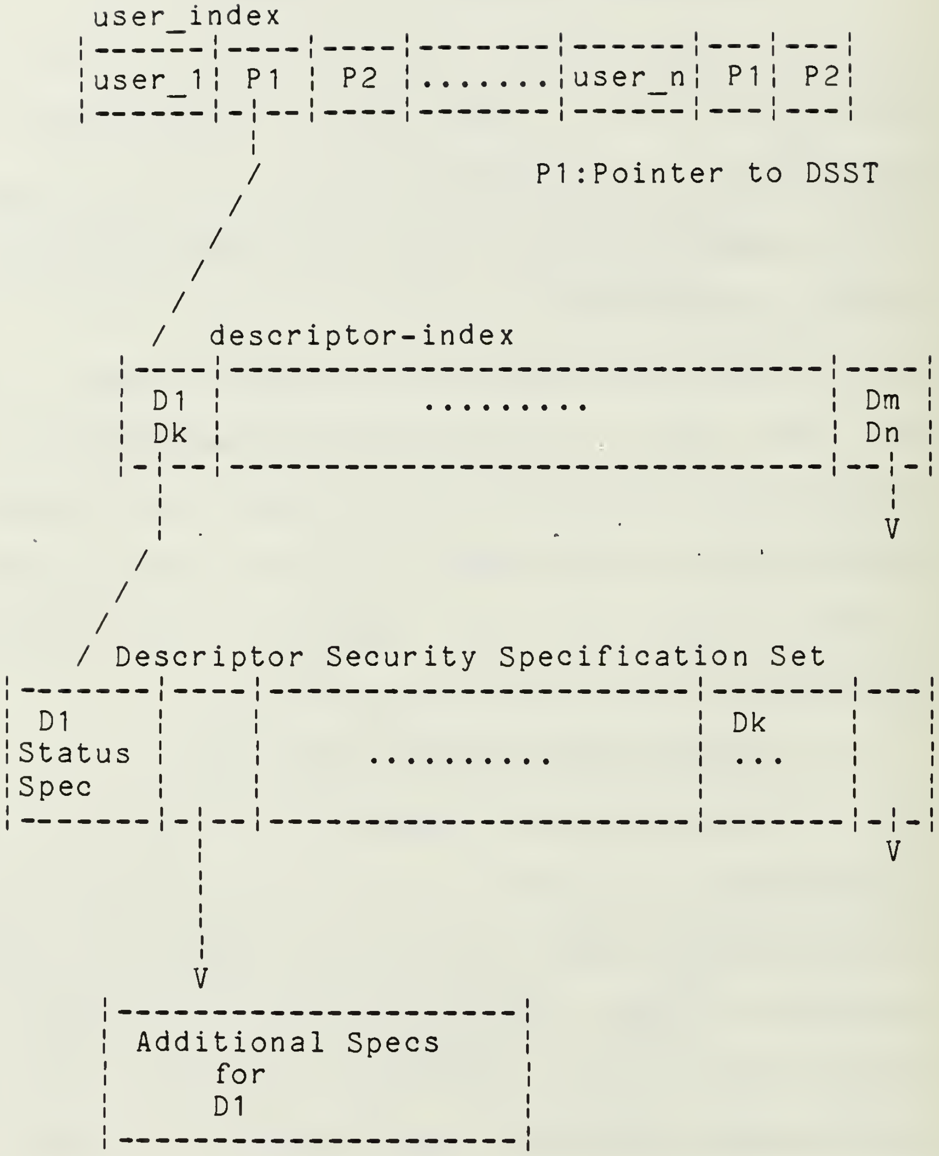


Figure 3.3 The Secondary-Storage-Based DSST

table in the backends' main memory. The user-index will have a pointer to the DSST entry for each user.

In the DSST, we deal with the descriptor-ids rather than the descriptor boundary values. This gives us the opportunity to use a simple layered indexing technique with variable length security specification entries.

4. The Cluster-level Access Control Information

After the database creator has defined the descriptors and the descritor-level access control information, then the database loading mechanism can create the clusters and the cluster-level access control information.

A cluster has been defined by a set of descriptors. The cluster-based security specifications are the union of the corresponding descriptor-based security specifications in the cluster. Consequently, the descriptor-based security specifications and the cluster-based security specifications can be represented in the same way. If the cluster is defined by only one descriptor, then the cluster-based security specification and the descriptor-based security specification will be the same. If the cluster is defined by more than one descriptor, then the union of the descriptor-based security specifications will be the cluster-based security specification.

Let us illustrate how to obtain the cluster-level access control information from a set of descriptor-based security specifications. Consider the database with four employee records depicted in Figure 3.4. The database creator first specifies some of the attributes as the directory attributes and then specifies some descriptors on the basis of these attributes. This information is stored in the DDIT (Figure 3.5). In this example, the attributes, Salary and Department, have been specified as the directory attributes. The attribute of the descriptors D1 and D2 is Salary and descriptors D3 and D4 is Department. In addition, the database creator defines descriptor-level access control information on each descriptor for every user in the database. This information is stored in the DSST (Figure 3.6). Now, the database creator has provided necessary information for the directory management and the access control mechanism. Using the entries and the information in the DDIT, MDBS can form clusters and this information is stored in the CDT (Figure 3.7).

With the descriptors corresponding to a cluster (in CDT) and the descriptor-based security specifications corresponding to the descriptors (in DSST), we can determine those security specifications corresponding to the cluster and store them in the cluster-to-cluster-based-security-specification table (CSST)(See Figure 3.8). For example in Figure 3.8, the cluster-1 is defined by the descriptors D1 and D3. In Figure 3.6, the descriptor-based security specifications on the descriptors D1 and D3 for the user-1

Record No	Attributes			
	Employee	Department_no	Salary	Manager
1	12	1	1000	12
2	13	1	1500	12
3	14	2	1200	14
4	15	2	2000	14

Figure 3.4 A Sample Database with four Employee Records.

Descriptor_id	Descriptor
D1	1500 <= Salary
D2	1500 < Salary < -
D3	Department_1
D4	Department_2

Figure 3.5 Descriptor-to-Descriptor_Id Table for the Database of Figure 3.4.

Descr_id	Descriptor-ss for user_1	Descriptor-s for user_2
D1	NONE	NONE
D2	{<Avg,Sal,No_Retrieve>}	{<Avg,Sal,No_Retrieve>}
D3	NONE	{<--,Name,No_Update>}
D4	{<--,Name,No_Delete>}	NONE

Figure 3.6 A Descriptor-to-Security-Specification Table (DSST)

Cluster_id	Set of descriptors	records in the cluster
1	{ D1,D3 }	R1,R2
2	{ D1,D4 }	R3
3	{ D2,D3 }	--
4	{ D2,D4 }	R4

Figure 3.7 The Cluster Definition Table (CDT) for the database of Figure 3.4.

Clus id	Cluster-based Security Specifications for user-1	Cluster-based Security Specifications for user-2
1	NONE	{<--,Name,No_Update>}
2	{<--,Name,No_Delete>}	NONE
3	{<Avg,Salary,No_Retrieve>}	{<Avg,Salary,No_Retrieve>, <__,Name,No_Update>}
4	{<Avg,Salary,No_Retrieve>, <--,Name,No_Delete>}	{<Avg,Salary,No_Retrieve>}

Figure 3.8 Cluster-to-Cluster-Security-Specification Table (CSST) for the databese of Figure 3.4.

are 'none'. Consequently, the cluster-based security specification on the cluster-1 for user_1 will be 'none'. User_2 has descriptor-based security specifications as 'none' for descriptor D1 and {<--,Name,No_Update>} for descriptor D3. Consequently, the cluster-based security specification on cluster-1 for user_2 will be {<--,Name,No_Update>}. By repeating this process, for every cluster and every user in the database, the cluster-level access control information can be determined.

5. The Cluster-to-Cluster-Based-Security Specification Table (CSST)

The cluster-level access control information is stored in an augmented_CDT which is the CDT with an additional column for user-cluster-based security specifications. Each user has an augmented CDT. The augmented CDT is used in the cluster search, access control and address generation phases of directory management.

In the cluster search phase, the cluster-ids for those clusters whose records may satisfy a request are determined. In the access control phase, the cluster-ids whose records are not authorized for the user are determined and the unauthorized cluster-ids are eliminated. In the address generation phase, the secondary storage addresses of the authorized clusters are determined.

In [Ref.6], the access control information has not been considered. The CDT, without access control information, has been implemented as two secondary-storage-based tables. The first table is the descriptor-to-descriptor-id-bit-map table (DCBMT). The second table is the cluster-id-to-secondary-storage-address table (CSSAT). DCBMT is used for the cluster search phase and CSSAT is used for the address generation phase. We will store the access control information of CDT in a third secondary-storage-based table called the cluster-to-cluster-based-security-specification table (CSST).

6. The Secondary-Storage-Based CSST

Since each user needs only one CSST, we will need to store CSSTs in the secondary storage. We will store the user-index in the main memory of the backends and we will have one pointer for each user's CSST table(Figure 3.9).

CSST utilizes the cluster-ids for indexing. The user-CSST pointer leads to the cluster-index for that user. The cluster-index cells have upper and lower cluster-id value. We look for the cluster-index cells whose values cover the cluster-id in consideration. Each cluster-index has a pointer which leads us to the cluster-based-security-specification set. The cluster-level access control information is stored in the corresponding cluster-based-security-specification set cells. Each entry in the set has space for a certain number of field-level access controls. If there are two many field-level access controls, then some will be stored in an overflow block which can be found by

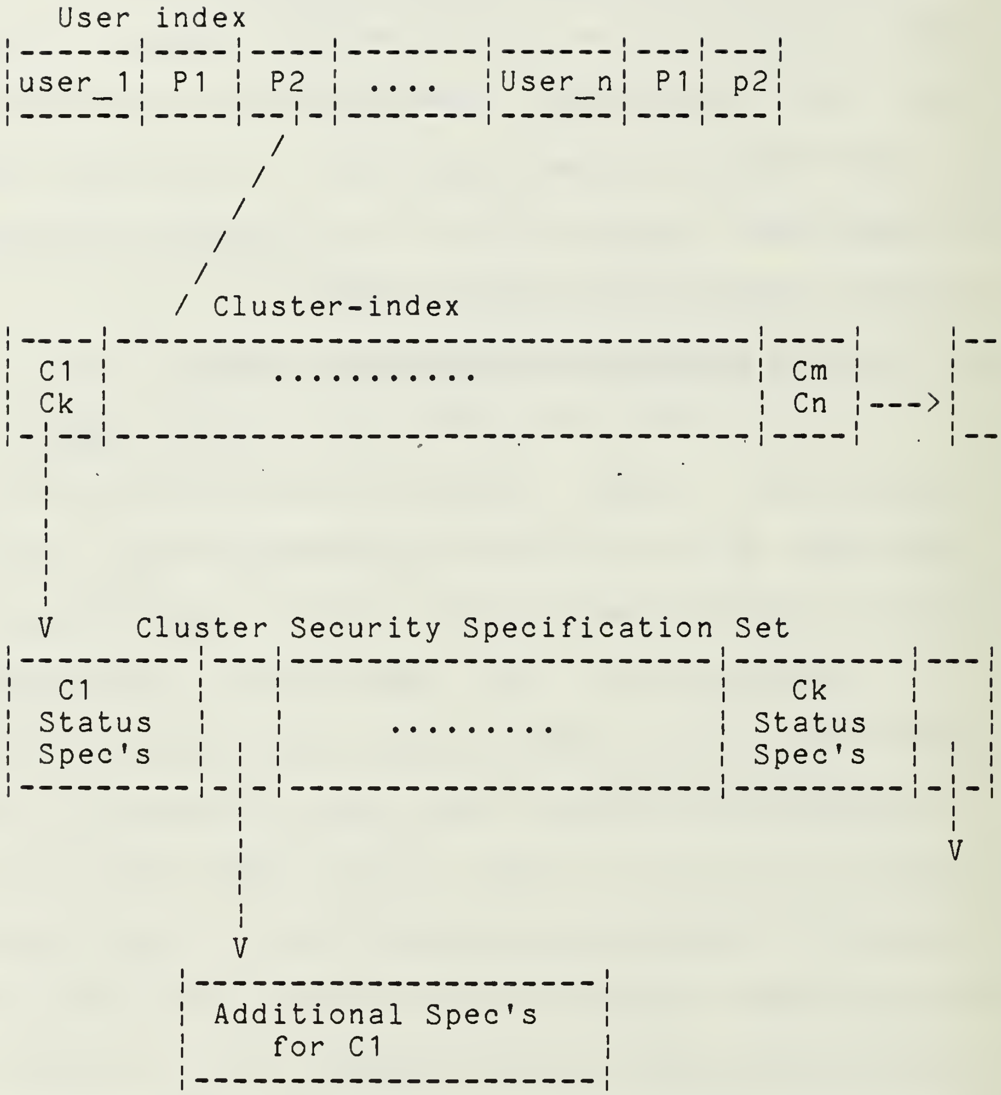


Figure 3.9 The Secondary-Storage-Based CSST

following a pointer from the cluster-based-security-specification set.

IV. PREPROCESSING

In this chapter, we will describe in more detail the preprocessing of the requests utilizing the access control information. As in the other phases of directory management, the access control mechanism distinguishes between insert requests and non-insert requests (retrieve,update and delete). The difference between an insert request and a non-insert request is that the insert request deals with only one cluster at a time while the non-insert request may deal with many clusters at the same time. In the following two sections, we will, in turn, describe the preprocessing for non-insert and insert requests.

A. PRE-PROCESSING FOR NON-INSERT REQUESTS

Directory management with access control has four phases; descriptor search, cluster search, access control and address generation. Directory management starts with descriptor search which finds the descriptor-id groups for the request. Then cluster search finds the cluster-ids for the corresponding descriptor-id groups. When all cluster-ids are obtained for the request, access control checks are made in order to produce a list of authorized cluster-ids.

Since access control for the non-insert requests works on clusters, we can utilize the cluster-level access control information for the request authorization. The request authorization is performed in two steps. The first step is cluster security specification search and the second step is cluster authorization decision. In the following two subsections, we will elaborate these two request authorization steps.

1. Cluster Security Specification Search

Cluster security specification search for a non-insert request begins with a list of cluster-ids for clusters to be authorized. We will use the cluster-based security specifications, stored in the CSST (Figure 3.9), as the access control information. CSST has three layers of indexing; user-index, cluster-index and cluster security specification set (cs-set). The user-index may be stored in the main memory, while the rest of the table is stored in the secondary storage. An algorithm to search CSST is as follows.

After the cluster search phase, for the cluster-ids found, the access control phase is started. First, the access control mechanism refers to the backend's main memory and finds the particular user-id in the user-index. The user-index shows us whether the user can access the database or not. If the user is not authorized to access the database, then the request is rejected. If the user is authorized to access the database, then a pointer to the cluster-index is fetched from the user-index. Thus, the first layer indexing (the user-index) allows us to check the user-access rights to the database.

With the pointer found in the user-index, the first block of the cluster-index is fetched from the secondary storage. Each cluster-index block has cells which have one upper and one lower bound cluster-id, and one pointer. If a cluster-id is between the upper and lower bounds of a cell, then we say that the cell covers this cluster-id. The pointer in the cell leads us to the cs-set which has its clusters covered by the corresponding cell. For example, in Figure 3.9, the first cell of the cluster-index has lower bound as C1 and upper bound as Ck. Consequently, The cluster-ids between C1 and Ck are covered by this cell and the pointer in this cell leads us to the corresponding cs-set block for these clusters. The above procedure is repeated for each cluster-id yielding a pointer to the cs-set for each of the corresponding clusters.

It can be observed that the strategy to search CSST is to search it one layer at a time. For each block fetched, the corresponding operations are performed for all clusters. This strategy prevents redundant accesses to the CSST blocks.

The next step is to search all the cs-sets. Recall that each cs-set block stores the access control information for a certain number of clusters. Each cluster in the cs-set has spaces for a part of the security specification which is a certain number of field-level access controls. If a cluster has more field-level access controls, then they are stored in an overflow block which can be reached with a pointer.

The cs-set blocks are searched one at a time, starting with the cs-set block for the cluster with the smallest cluster-id. A cs-set block is searched for all clusters we are dealing with. After the processing of the cs-sets are completed for all clusters, additional field-level access controls for clusters are fetched from overflow blocks utilizing the pointer obtained in the cs-set.

2. The Cluster Authorization Decision

The cluster-based security specification for each cluster to be authorized has been provided from CSST in the cluster security specification search. Now, the access control mechanism should utilize the user request and the cluster-based security specifications in order to decide about the cluster authorization for the request.

In [Ref.3], the relationships among the disallowed access operations in order to prevent compromising the access control mechanism are described. These relationships, called the <u>access control implications</u>, are as follows:

```
No_Retrieve ---> No_Update
No_Retrieve ---> No_Delete
No_Update   ---> No_Delete
No_Delete   ---> No_Update
```

The determination of the security specification for each cluster to be authorized is the same for all types of non-insert requests. However, the processing of this information for each type of request is, of course, different. This processing will be described in the next three subsections.

a. The Authorization for Retrieve Requests

The syntax of a retrieve request is:

```
RETRIEVE Query Target-List [BY Attribute]
```

Where the query is any arbitrary Boolean expression of predicates which identifies the portion of the database to be retrieved. The target-list is a list of elements. Each element is either an attribute, e.g., SALARY, or an aggregate operator to be performed on an attribute, e.g., AVG(SALARY). We support five aggregate operators - AVG, SUM, COUNT, MAX, MIN - in MDBS. Let us have an example using the database in Figure 3.4. The retrieve request could be as follows.

```
RETRIEVE (File=Employee) and (Department=2) (Avg(Salary))
```

The meaning of the above request is to retrieve the average salary of the employees who are in department 2. Let us assume that directory management has performed descriptor search and cluster search. Determining that clusters C2 and C4 may contain records satisfying the request. The access control mechanism performs the cluster security specification search for user-1 as described in the previous section and finds the security specifications from the CSST (see Figure 3.8) as follows:

(C2) {<---,---,No_Delete>}

(C4) {<Avg,Salary,No_Retrieve>,<---,---,No_Delete>}

The request calls for retrieving the salaries of the records in cluster C2 and cluster C4. User-1 does not have any contradictory security specification for retrieval of cluster C2, since cluster C2 has a security specification which only restricts the deletion of records. This check operation is performed as follows. The access control mechanism compares the attribute in the security specification, which is null, and the attribute in the target-list part of the request (i.e. salary). Since they do not match, it authorizes the cluster. In the case of cluster C4, the security specification has two field-level access controls. Let us consider the first field-level access control. The aggregate operator of both the field-level access control and the request match (i.e. average). In addition, the attribute of both the field-level access control and the request match. Now, the access control mechanism checks the disallowed access operation. Since it is No_Retrieve, the request is not authorized to retrieve the records of cluster C4. As a result only the cluster C2 is authorized for this request and the user is warned as only authorized records have been retrieved.

Let us now consider a slightly different request as follows:

```
RETRIEVE (File=Employee) and (Department=2) (Salary)
```

The query in the request is the same as in the previous example. However, the target-list calls for retrieving the salaries of particular employees instead of the average salary. Since the query is the same as in the previous example, clusters C2 and C4 will again be the satisfying clusters. In the first field-level access control for cluster C4, we see that the retrieval of the average salary is not authorized. But here we want to retrieve only the individual salaries. Now, we will have an access control implication: if a request with an aggregate operator is not authorized, then the same request without aggregate operator will also not be authorized. Thus, for example, if the retrieval of the average salary is not authorized, then the retrieval of the individual salaries is also not authorized. The inverse of this implication is not true. That is, even if we have restriction the retrieval of salaries, retrieval of the average salary can be authorized. We can state this as follows.

<Avg,Salary,No_Retrieve> ---> <---,Salary,No_Retrieve>

<---,Salary,No_Retrieve> -/-> <Avg,Salary,No_Retrieve>

We can add this rule to the access control implications as the fifth access control implication. We should recall that the aggregate operations are used only in retrieve requests. The fifth access control implication can be stated more formally as follows.

<Agg-op,Attr-A,access op-B> ---> <Agg-op,Attr-A,access op-B>

b. The Authorization for Delete Requests

The syntax of a delete request is:

DELETE Query

where the query specifies the particular records which will be deleted from the database. Let us consider an example for a delete request from user-1.

DELETE (File=Employee)and(Salary=1000)

User-1 wants to delete those employee records which have salary as $1000. The query has only one predicate which is covered by the descriptor D1 (see Figure 3.5).

Consequently, cluster C1 (defined by descriptor D1 and D3) and cluster C2 (defined by descriptor D1 and D4) contain records which may satisfy the request since they have records with salary less than or equal to $1500.

The cluster security specification search of the CSST shown in Figure 3.8 finds the security specifications of clusters C1 and C2 for user-1 are as follows:

(C1) NONE

(C2) {<---, ---, No_Delete>}

Cluster C1 does not have any security related restriction. Consequently, we can reach the records R1 and R2 (See CDT in Figure 3.7). We should recall that cluster C1 contains those records which have salary less than or equal to $1500; but, the request needs only those records with salary equal to $1000, in this case record R1.

According to the security specification for cluster C2, the user is not authorized to delete a record in cluster C2. Consequently, the delete request for cluster C2 is rejected. In addition, according to the access control implications, even if the dissallowed access operation were No_Retrieve or No_Update, then cluster C2 would be unauthorized. As a result, only the record R1 in cluster C1 will be deleted.

c. The Authorization for Update Requests

The syntax of an update request is:

```
UPDATE Query Modifier
```

where the query specifies the records to be updated and the modifier specifies the update operation. Five basic modifier types have been specified in [ref.4] and briefly described in chapter 2. Each of the modifier types try to alter the value of one attribute. Since the security specifications specify the disallowed access operation for the attributes, we can have an authorization decision immediately after we find out the attribute to be modified.

Let us consider an example of an update by user-2.

```
UPDATE (File=Employee)and(Dept=2)and(Salary>1500)
       <Salary=Salary+50>
```

User-2, with this request, wants to increase by $50 the salary of those employees who are in Department 2 and have salary greater or equal to $1500. Cluster search determines the satisfying cluster as only cluster C2, which is defined by descriptors D2 and D4. Cluster C2 has the security specification with only one field-level access control for user-2 as:

{<--,salary, No_Update>}

The access control mechanism compares the attribute in the each field-level access control of the security specification (here Salary) and the attribute in the modifier (here again Salary). If they did not match, then the cluster would be authorized. Since they match, the access control mechanism looks at the dissallowed access operation. Since the dissallowed access operation is No_Update, the cluster is not authorized. Here, we have only one satisfying cluster for the query and this cluster is unauthorized. Thus, the request will be rejected. If the dissallowed access operation were No_Retrieve or No_Delete, then the request would also be rejected because of the access control implication rules.

In the previous sections, we have described the preprocessing for non-insert requests. In the following section, we will describe the preprocessing for insert requests.

B. PREPROCESSING FOR THE INSERT REQUESTS

The MDBS data manipulation language specifies the insert request as follows:

INSERT <Request>

where the Request is of the form {<Attribute,Value>, <-->, ..., <-->}. As stated previously an insert request differs from non-insert requests because it refers to a single unique cluster. The actual processing of an insert request depends on whether or not this cluster already exists. In the following subsections, we will describe the preprocessing of an insert request in detail using the following example:

INSERT {<File,Employee>,<Salary,1250>,<Dept,TOY>}

1. Insert Requests for an Existing Cluster

In the case of an insert request, descriptor search may be performed by more than one backend, but the actual insertion will be done by only one backend. If the cluster already exists, i.e., already contains other records, then the backend will also have the cluster-level access control information in its CSST. The access control process for the insert request with an existing cluster is performed only by the one backend which is to perform the actual insertion operation. The access control mechanism in this backend handles this kind of insert request the same as a non-insert request with one cluster.

At the end of the descriptor search, the satisfying descriptor-id group is determined. The cluster-id search will determine the unique cluster-id from CDT for the descriptor-id group. The backend, chosen for the insertion, will perform the cluster security specification search on this cluster-id. This search is the same as the cluster security specification search described for a non-insert request. Having been provided the security specification for the cluster and the user-id, the access control mechanism will check the authorization on the request. If the security specification does not have any disallowed access operation as No_Insert, then the request will be authorized.

Let us now consider the example insert request for the database in Figure 3.4. The user wants to insert an employee record for an employee, who will have salary of $1250 and who will be in department 1. In descriptor search, the satisfying descriptor on attribute Salary is found as descriptor D1, since descriptor D1 covers all of the salaries which are less than or equal to $1500 (see Figure 3.5). The descriptor for the attribute department is found as descriptor D3. Thus, the descriptor-id group for this request is descriptors D1 and D3. In the cluster-id search, directory management searches the CDT (see Figure 3.7) for the descriptor-id group and the cluster is found to be cluster C1. After this point, processing is the same as the access control phase of non-insert requests. The security specification for cluster-1 will be searched from the CSST and the security specification will be examined for authorization of the request.

2. Insert Requests into the Empty Cluster

A second case occurs if all of the descriptors for the insert are already defined, but this insert is the first one for the cluster defined by this descriptor-id group. In this case, no information about the cluster can be found in the directory. Consequently, we can not find any cluster-based security specifications for this cluster. However, since all of the descriptors are the existing descriptors, we can find the descriptor-level access control information in DSST. Thus, we can derive the cluster-level access control information from the descriptor-level access control information as described in the algorithm in section III-D-4.

The descriptor-based security specifications for each descriptor are fetched from DSST. The union of the descriptor-based security specifications is stored in CSST as the cluster-based security specification for this cluster. After this point, the authorization step for the request is performed as before. Thus, for example, if one of the disallowed access operations in the cluster-based security specification is No_Insert, then the request is rejected.

The method described above has an efficiency problem. We derive the cluster-based security specification, store it in CSST and then perform the authorization decision. If the insertion is authorized, then the record is inserted. If it is not authorized, then the insertion is rejected. In this case, the effort to update the CSST has been performed even though no insert actually occurs. We should find a better solution to overcome this inefficiency.

We reject an insert request if the dissallowed access operation in a descriptor-based security specification is No_Insert. During the DSST search, we can check the dissallowed access operation for each descriptor. If we find a dissallowed access operation of No_Insert in a descriptor-based security specification, then the request can be rejected immediately. In this case no update to the CSST will be performed. If none of the descriptor-based security specifications has the disallowed access operation of No_Insert, then the request is authorized and the CSST can be updated accordingly.

3. Insert Requests into a new Cluster

A third case arises if the request needs to create a new descriptor. In this case not only will the cluster not exist, but there will be no descriptor-level access control information for this new descriptor. Thus, of course, we can not derive the cluster-level access control information as was described in the last section.

Insert requests with a new descriptor introduce two new problems. The first is 'Who can create a new descriptor?' The second is 'How can the descriptor-based security specification for the new descriptor be specified?'. The first question must be answered prior to the actual request authorization. Consequently, the access control process for the insert request with a new descriptor can be divided into three phases; the new descriptor creation authorization, the access control information specification for the new descriptor and the total request authorization.

In the initial access control design in [Ref.3], the problem created by a new descriptor has not been addressed. We will analyze this problem further in the following chapter.

V. INSERT REQUESTS WITH THE NEW DESCRIPTORDS

In chapter 4, we described the preprocessing for insert and non-insert requests utilizing the cluster-level and the descriptor-level access control information developed in chapter 3. At the end of chapter 4, we described an insert request with a new descriptor and observed that there was no access control information at the cluster-level or the descriptor-level for this kind of request. In this chapter, we will describe the access control information and the access control methods for insert requests with a new descriptor. Let us look first at what the reason is to have a new descriptor.

A. THE REASON FOR THE NEW DESCRIPTOR

The database creator specifies the descriptors for Type A and Type B attributes as an upper limit and a lower limit on the attribute value. Thus, we never need to create a Type A or Type B descriptor, since the value of the predicate attribute will fall within the boundaries of the descriptors. However, if the attribute is of Type C, then each value ot the Type C attribute will result in a distinct descriptor. For example, if attribute Department is of Type C, then Department TOY and Department SALES are the distinct descriptors. Since each value of the Type C attribute results in a Type C descriptor, there may be some values which are not defined as descriptors by the database creator. For example, if Department ADVERTISING has not been defined earlier but we want to create a new department as ADVERTISING then we create a new Type C descriptor on the Type C attribute department. We may even need to create more than one Type C descriptor for an insert request.

B. THE AUTHORIZATION REQUIREMENTS FOR A REQUEST WITH A NEW DESCRIPTOR

In the directory management processing without access control, the new descriptors are created and broadcasted by the controller. This process is done as follows. In the descriptor search phase, if the backend can not find the descriptor for a predicate, then it looks at the type of the predicate attribute. If the attribute is of Type C and the request is an insert request, then it sends a message to the controller to create a new Type C descriptor. The controller creates a new descriptor and broadcasts it to the all backends. The backends receive the new descriptor and store it into the proper tables.

Directory management with access control requires us to add two more steps to the process described above. The first step is 'the authorization step for the new descriptor' and the second is 'the descriptor-level access control specification for the new descriptor'. What will be the order for these steps? New descriptor creation is a costly process in terms of the number of messages among the backends and the controller. Thus, we will try to to avoid the unnecessary creation of a descriptor. Obviously, the new descriptor authorization checks will be performed before the new descriptor creation. Total request authorization also depends on the other descriptors' security specifications as well, we should refer to DSST to check the security specifications for all of the descriptors in the cluster.

Another necessary operation is to specify some descriptor-based security specifications for the new descriptor. The new descriptor's security specification is specified and stored in DSST, and the cluster-based security specification is derived from all of the descriptors' security specifications.

Now, we know what we should do for the insert request with a new descriptor. The order of the operations and the methods for the operations must still be determined. In the following sections, we will elaborate the steps of the process in order to see and assess the various choices. These steps are the new descriptor-creation authorization, the new descriptor creation operation, the descriptor-level access control specification for the new descriptor, the authorization decision for the old descriptors and the cluster-level access control information derivation for the new cluster.

C. THE NEW DESCRIPTOR CREATION AUTHORIZATION

We know that each value of a Type C attribute is considered for a distinct Type C sub-descriptor. If department and job are specified as Type C attributes, then each values of one of these attributes is also the value of a distinct Type C sub-descriptor. The creation of a new department defines a new descriptor. Another example can be to add a new job to the system. The access control question for this kind of operation can be stated more formally as 'who is the person allowed to create a new descriptor on this attribute?'. This question can be answered by the database creator at the database creation time, since it depends on the basic organizational schema.

In MDBS, deletion and creation of an attribute is not allowed after the database is first created. Since the attributes are permanent, we can give some security specifications on the attribute-level for the future descriptor-creation authorizations. These security specifications will be on Type C attributes for each user in the database. We store the user-descriptor-creation authorizations in a New-Descriptor-Authorization Table (NDAT).

1. The New-Descriptor-Authorization Table (NDAT)

We store the descriptor-creation authorizations for each user on the Type C attributes in this table (Figure 5.1). The descriptor-creation authorization on each Type C attribute for user i is a boolean expression. In Figure 5.1, we can see that creation of a new department is authorized for user 2 but not for user 1.

2. The Secondary Memory Based NDAT

We may want to store NDAT in the secondary storage, since the dimension of NDAT depends on the number of the users in the database which may be large. Figure 5.2 shows a NDAT with secondary-based implementation.

The entries of NDAT are the user-ids. Only the authorized Type C attributes for the new descriptors are listed in NDAT. Thus, each user will have a variable length entry with the attribute-ids in it. We may not want to use well-structured indexed or B-Tree constructions for this table, since the number of the Type C attributes in the database is not large. In addition, most of the users will have creation authorization only for a subset of these attributes. In Figure 5.2, we can see that user 1 can create some new descriptors on the attributes which are listed. However user n is not authorized to create any descriptors.

User_1	P1	P2	P3	.....	User_n	....

Attr ID	User_1 Creation Authorization	User_2 Creation Authorization	....	User_n Creation Authorization
Dept	NO	YES	......	
Job	YES	NO	......	
..	......	....	......	

Figure 5.1 New Descriptor Authorization Table (NDAT)

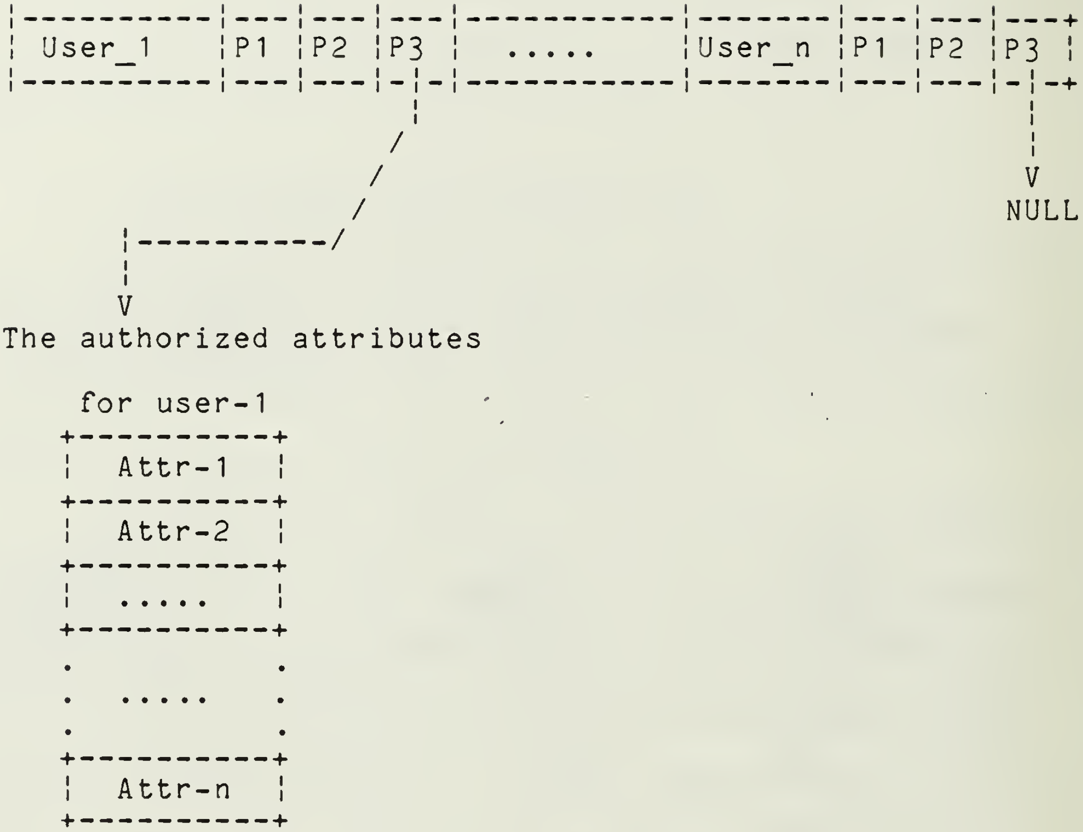


Figure 5.2 The Secondary-Memory-Based Implementation of New Descriptor Authorization Table (NDAT)

D. NEW DESCRIPTOR CREATION TIME

We know that the request authorization for the insert requests with the new descriptor depends on either the new descriptor-creation authorization or the access authorization for the old descriptors. The new descriptor-creation authorization can be performed before the creation. If we create a new descriptor and the request is denied because of the old descriptors' security specifications, then we will have an unused, unnecessary descriptor.

Let us consider the following insert request for user 1:

```
INSERT {<Salary,2000>,<Job,SALESMAN>,<Dept,TOY>}
```

where both attribute Department and attribute Job are Type C attributes and the predicate values, TOY and SALESMAN, do not exist as descriptors. Thus, new Type C descriptors should be created, TOY for the attribute Department, and SALESMAN for the attribute Job, respectively.

Assume there are three or more backends and descriptor search for predicate 1,2 and 3 will be performed by backend 1,2 and 3, respectively. Thus, backend 1 performs descriptor search on the attribute Salary and finds the corresponding descriptor. On the other hand, backend 2 can not find descriptors for attribute Job with the value SALESMAN. Backend 2 looks at the type of the attribute Job and finds out that it is a Type C attribute. Consequently, backend 2 wants to create a new Job as SALESMAN. Now, backend 2 has to check whether or not user 1 is authorized to create a new job. Backend 2 performs the new descriptor-creation-authorization checks for user 1 on attribute Job. This operation is performed as follows. The NDAT block for user 1 is fetched from the secondary memory via the pointer provided in the user-index for user 1. We search the block to find the attribute Job. Since Job is in the block, we conclude that user 1 is authorized to create a new job. Thus, backend 2 requests a new descriptor, namely SALESMAN, from the controller. The controller creates a new descriptor and broadcasts it to all backends.

Similarly, backend 3 performs descriptor search on the attribute Department and finds out that no descriptor exists for TOY. It performs the new descriptor-creation-authorization check on attribute Department for user 1. Since attribute Department is not indicated in the NDAT block for user 1, user 1 is not authorized to create a new department. The request must be totaly rejected. Now, we have created a descriptor, namely SALESMAN, but we could not insert the record. Thus, we have a descriptor for a name not in the database. In addition, we have wasted an effort for the new descriptor creation which is costly. The same problem would also arise with predicate 1. If the user does not have an access right for insertion of the records which have salary of $2000. Some possible strategies to avoid this problem are described in following sections.

1. Method-1 for new Descriptor Creation

This method tries to create the new descriptor as the last step. Each backend first tries to decide about the creation authorization for the new descriptor and the access authorization for the old descriptors. If everything is authorized, then the backend sends a status message to the other backends. After receiving the status messages from all the backends, we can conclude that the request is authorized and we can demand new descriptors. The controller creates the new descriptors and broadcasts them to the backends.

2. Method-2 for new Descriptor Creation

The problem with method-1 is efficiency, since each backend has to wait for all status messages before requesting a new descriptor creation. Method-2 allows the backends to create a new descriptor whenever it is needed, and if the request is denied, then the new descriptors are deleted.

The new descriptor deletion operation is requested by the creator backend from the controller. The controller deletes the new descriptor and broadcasts a deletion message to the backends.

This method is also inefficienct. In the worst case, if there are N backends in the system, then we need N additional messages from the backends for deletion and one broadcast message from the controller. In addition, we will have some deletion overhead in the controller.

3. Method-3 for new Descriptor Creation

Method-3 also allows the backends to create a new descriptor whenever it is needed. The controller creates a new descriptor by assigning a descriptor-id and it does not keep track of the value of descriptors. Thus, if the backends do not store the new descriptors into DDIT and into other tables, then there is no way to reach them again. If the backend needs the same descriptor for a later request, then the controller will assign another descriptor-id for this descriptor regardless of its prior existence. The only problem for this method is an increase of the number of descriptor-ids because of some nonexistent descriptors.

4. The Sequence of an Insert Request with the Superior Method

Method-1 is superior to others in terms of the uniform handling of the problem, the lack of side effects (e.g., the artificial descriptor-id increase seen in method-3) and relatively modest overhead. Let us elaborate method-1 and see the complete sequence for an insert request.

Directory Management receives the request and performs AT search for each predicate assigned to that backend. If there are Type C attribute in the request, then they are locked, and descriptor search is performed for each predicate, as follows. For each predicate, the backend reads the DDIT, finds the attribute and the proper descriptors on this attribute. If the attribute type is Type C and there is no descriptor for this attribute with this particular predicate value, then we need to create a new Type_C descriptor with this predicate value. The backend looks at NDAT in order to check "creation authorization" for the user on this predicate attribute. If the user is authorized to create a new descriptor on this attribute, then the backend continues descriptor search for the next predicate. If the user is not authorized to create a new descriptor on this attribute, then we deny the whole request.

After descriptor search is done for all predicates, the backend broadcasts them and waits to get descriptor-ids from the other backends. With this broadcast message, the backend sends a flag indicating that a new descriptor is to be created.

The processing for any type of insert request is the same until this point. There may be two cases at this point. The first case is that no backend needs to create a new descriptor, i.e., this is an insert request with an existing or empty cluster. The second case is that one or more backends need to create a new descriptor. We have elaborated the first case in the chapter 3. Let us consider the sequence of processing for the insert request with a new descriptor after this point.

The backends start "descriptor-level access control process" on the descriptors that they have broadcasted. There is no cluster for this request available to be inserted into and to be checked for cluster-level access control information. We have not yet created the new descriptor. We delay the creation until after the descriptor level access control process.

Let us look at the descriptor-level access control process, which is the same as for an insert request with the empty cluster. There is only one descriptor-id group for the request since insert requests have only one cluster. If we check user authorization for each descriptor and each of the descriptors contains positive insert authorizations, then we can deduce that the request is authorized. This process is performed by all of the backends. After positive authorization decision is completed, the backends which need new descriptors request new descriptors from the controller. When the controller sends a new descriptor creation message, the backends start to store this new desriptor in the proper tables, and they create a new cluster and the cluster-level access control information.

E. THE ACCESS CONTROL INFORMATION SPECIFICATION FOR THE NEW DESCRIPTORS

We still have to find an answer for the question "Who can access to a cluster which is defined by this new descriptor"? For example, if the new descriptor is a new department, namely, TOY, then the question will be 'who can read the salaries of the employees in this department and who can not?'. Can we decide about this specification in advance? Let us elaborate some choices for answers to this question.

1. Method 1 for the Access Control Specification for a new Descriptor

This method requires the database creator to specify all security specifications at the database creation time. We have already decided that the new Type C-descriptor-creation authorizations for each user would be specified at the database creation time. Now, we have to ask 'can we also specify the user access authorizations on this new descriptor at the database creation time?'. In some cases it may be possible. For example, the director of a department will have right to read 'new employee's salary'. This is very appropriate in terms of a static security policy. However, we may not be able to predict all necessary security specifications for new descriptors in advance. For example, suppose we want to create a new department which is a totally new managerial decision. It is possible that it does not fit into the regular policies specified in advance. Thus, we will need some additional security specifications. Thus, the solution for this problem may be to specify clear and available security specifications at the database creation time and to use the database administrator/creator's security specification facilities to change or add new security specifications later.

2. Method 2 for the Access Control Specification for a new Descriptor

It is clear that we may not be able to specify all security specifications for some specific descriptors at the database creation time. If we have the flexibility to specify some security specifications after descriptor creation time, then we will have an additional problem of dealing with an insecure gap between the descriptor creation time and new access control information specification time. We may not want to specify the new security specifications at the new descriptor creation time in the same insert request, since such a specification will be too slow for the insert requests. This gap must be covered by an additional security mechanism to prevent unauthorized accesses. One alternative is to 'assume full restriction for all users' at the beginning and than to insert some security specifications to authorize some users to access if such access is needed. In this way, the system will have no insecure gap and the system will be easy to handle in terms of the overhead.

3. A Comparison of the methods

Method 1 provides a more secure and practical security mechanism. With method 1, the overhead will appear at the time of insertion the new security specifications. Initially, we will isolate all users from the newly created descriptor and therefore from the new cluster. Thus, all of the cluster-based security specifications will be "all accesses are disallowed" for all of the users. We do not care about each descriptor's security specifications in the new cluster in order to derive the cluster-based security specification from the descriptor-level access control information.

If we want to authorize a user with an access right on the new descriptor, then we have to go back and check the security specifications for the other descriptors in the defining cluster. That is, we must determine if there are any descriptors with more strict security specifications. If one of the old security specifications is more strict than the new security specification, then we accept the old one and we warn the user. Let us consider the following example:

INSERT {<Salary,10000>,<Job,SALESMAN>,<Dept,TOY>}

Let us assume that the security specifications in the augmented_DSST for user_i on the descriptors in the satisfying cluster are as in Figure 5.3. Let the new cluster be defined by three descriptors where descriptor Dc is the new descriptor for SALESMAN. The default security specification for the new descriptor is 'all accesses are disallowed for user_i. The cluster-based security specification will be the union of the descriptor-based security specifications of the corresponding descriptors. Here, the security specification of descriptor Dc owerhelms the others and the cluster-based security specification for the new cluster for user_i will be 'all accesses are disallowed'.

If we want to authorize user_i to read those emloyees' salary who are SALESMAN, then we change descriptor Dc's security specification. The new security specification will have all possible accesses are disallowed except retrieval of the Salary attribute. After this descriptor-based security specification has been changed, we have to rederive the cluster-based security specification. For this process, we have to look at the DSST, the security specifications for descriptors Da and Db, in order to make sure that other descriptors in the cluster do not have any conflicting security specifications which would not

Descr id	Descriptor	Descriptor-based Security Specifications for user-1
Da	0<Salary<10K	<Name,No_Update>,<Salary,NO_Retrieve>
Db	Dept=TOY	<Dept,NO_Update>
Dc	Name=JOE	ALL

Figure 5.3 An Augmented DSST for User i

authorize user i to read SALESMEN's salary. In this example descriptor Da has the restriction for user_i, such that, user_i can not retrieve the SALESMAN's salary if the salary of the salesman is between 0 and 10000. Since the salary is in this range, the modified descriptor-level security specification will have no effect on the cluster-based security specification. i.e., descriptor Da's security specification (salary,No_Retrieve) is the most strict restriction. Thus, the cluster-based security specification will be unchanged as 'ALL'.

If descriptor Da did not have a restriction on the retrieval of salaries, then security specification derivation would be different. We would have specified a field-level access control as <salary,No_Update>, since we want user i to be able to retrieve the salaries of the employees but not to change them.

We have elaborated the directory management information based postprocessing in chapter 3,4 and 5. In the next chapter, we will elaborate postprocessing phase of the access control system.

VI. THE POSTPROCESSING

In chapter III, we described how access control was divided into preprocessing postprocessing, and we developed the database-creator-specified access control information for preprocessing access control. In chapter 4 and 5, we have elaborated the preprocessing access control. In this chapter, we will develope the access control tools of postprocessing. In the rest of the chapter, we will use the term 'postprocessing' for the postprocessing step of the access control whenever there is no confusing with the postprocessing function of the controller.

A. ACCESS CONTROLS IN POSTPROCESSING

We described the MDBS access control system mostly as preprocessing. Recall that preprocessing is restricted to the directory attributes. Now, we also want to establish a protection mechanism on the non-directory attributes. In preprocessing, we have been able to restrict some operations on the non-directory attributes. For example, the descriptor containing the attribute value TOY has been specified on the directory attribute Department. In addition, the attribute Salary is a non-directory attribute. Assuming that the descriptor-based security specification for TOY is: <--,Salary,No_Retrieve>. The meaning of the descriptor-based security specification above is: the user can not retrieve the salaries of those employees who are in the department TOY. Thus, we protect a non-directory attribute (Salary) from an operation (retrieve) with regard to the value of a directory attribute (department=TOY). The purpose of postprocessing is to establish protection on the directory or non-directory attribute with regard to the value of a non-directory attribute. If the attribute department in the previous example were a non-directory attribute, then the access control would require postprocessing instead of preprocessing.

B. THE POSTPROCESSING PHASE

We know that each backend in MDBS has three major functions: directory management, concurrency control and record processing. We also know that access control preprocessing is performed in directory management utilizing the directory management related information.

Postprocessing deals with the values of the non-directory attributes. There is no way to determine the physical location of the records with regard to their non-directory attribute values. Thus, we can perform the postprocessing only after record retrievals by the record processing function of the backends.

Record processing has two major functions: the physical data operations and the aggregate operations. A physical data operation depends on the type of the request. For insert requests, the physical data operation stores the records into the secondary storage. For non-insert requests, the physical data operation first fetches the records from the secondary storage and filters the records which do not satisfy the query in the user request. It then performs the intended non-insert operation such as the deletion of record, update of an attribute value of the record or extraction of some attribute values of the record for a retrieve request. An aggregate operation performs the partial aggregate operation for the backend.

In the physical data operation, access control entails the follwing: First, it will check that the user has any security specifications specified on the non-directory attributes. If the user does not have any security specifications on the non-directory attributes, then the access control will have no affect in record processing. Let us assume that the user has some security specifications on the non-directory attributes. During the physical data operation, the backend fetches the records one track at a time. Each record is checked to see whether it satisfies the query in the request or not. If the record does not satisfy the query, then it is ignored. After this step, postprocessing checks will be performed for each satisfying record. After postprocessing, the satisfying and authorized records are sent to the aggregate function of record processing. In the following sections, we will describe the access control information and the steps in postprocessing.

C. THE POSTPROCESSING ACCESS CONTROL INFORMATION AS SPECIFIED BY THE DATABASE CREATOR

In this section, we will describe the database-creator-specified access control information for postprocessing. We will also describe how to store this information. The main idea of the access control information for postprocessing is the same as the access control information described in Chapter III for preprocessing. We, again, want to answer the following question: <u>who</u> can perform <u>what</u> operations on <u>which</u> part of the database. Specifying the first two items in the question will be the same as for preprocessing. This information is obtained from the user-ids and the access operations, respectively. Specifying 'which' portion of the database depends on the security specifications on the non-directory attribute values given by the database creator.

The database creator specifies attribute-level access controls on the non-directory attributes. An <u>attribute-level access control</u> specified on a non-directory attribute is of the form:

<Lower,Upper,Aggregate op,attribute,disallowed access op>

where lower and upper are the domain limits on the attribute, the aggregate operator is one of the aggregate operators, the attribute is a directory or non-directory attribute and the dissallowed access operation is one of the set {No_Retrieve, No_Update, No_Delete, No_Insert}. The aggregate operator may be used only for the attribute-level access controls with disallowed access operation as No_Retrieve. The attribute entry is only included when the dissallowed access operation is No_Retrieve or No_Update. If the non-directory attribute which will be protected has single value instead of value range, then the upper value will not be specified.

The difference between the field-level access controls for preprocessing and the attribute-level access controls for postprocessing are as follows. The field-level access controls are built on the descriptors while the attribute-level access controls are built on the non-directory attributes. In addition, for the attribute-level access controls, the value boundary to be protected is specified with the upper and the lower bounds which indicate the value range on the non-directory attribute. The rest of the access controls are the same.

An <u>attribute security specification</u> may be 'all' or a collection of attribute-level access controls. An attribute security specification of 'all' indicates that all accesses are disallowed for the respective user. The attribute security specification on an attribute is the collection of attribute-level access controls for the attribute. It can be represented as follows:

```
(Attribute) {<lower,upper,Agg_op,Attr,disallowed access op>,
                <--->,......,<--->}
```

For example, we might have the attribute security specification on the non-directory attribute Salary for the various ranges of the Salary as

```
(Salary) {<1000,2000,--,Job,No_Update>,
          <1000,5000,--,--,No_Delete>,
          <4000,5000,--,--,No_Insert>}
```

The meaning of this attribute security specification is as follows: the user can not update the Job attribute of a record, if the record has a salary between $1000 and $2000. Nor can the user delete a record if the record has a salary between $1000 and $5000. In addition, the insertion of a record with Salary between $4000 and $5000 is not authorized.

We store the access control information for the postprocessing phase into the two layers of tables, the non-directory-attribute table (NAT) and the attribute-security-specification table (ASST). These tables and their implementations are discussed in the next two subsections.

1. The Non-Directory-Attribute Table (NAT)

There is one NAT (see Figure 6.1), if the user has some restrictions on the non-directory attributes. In the NAT, we store only those non-directory attributes which have an attribute security specifications specified on them. The user-index in the backends' main memory has a pointer for the NAT for each user. If the user does not have any restrictions with regard to the non-directory attribute values, then this pointer will be null. The NAT corresponds to the attribute table in directory management. The secondary storage-based NAT implementation is also the same as the attribute table [Ref.6]. Thus it will not be elaborated here. Each attribute in the NAT has a pointer to the corresponding ASST for the attribute. The ASST is described in the next subsection.

2. The Attribute-Security-Specification Table (ASST)

We store the attribute security specifications for a particular non-directory attribute in an ASST (Figure 6.2). Thus, each non-directory attribute in the NAT has an ASST. The relationship between the NAT and the ASST is shown in Figure 6.3. We store the ASST in the secondary storage.

D. THE POSTPROCESSING OPERATION

The output of directory management is the secondary memory addresses of those records which are authorized by preprocessing. Those records are fetched and field-level

```
+-----+-------------------+----------+
¦Next ¦   Non-Directory   ¦Pointer to¦
¦Attr ¦     Attribute     ¦   ASST   ¦
+-----+-------------------+----------+
¦     ¦    Attribute A    ¦      ----+------->
+-----+-------------------+----------+
¦     ¦    Attribute B    ¦      ----+------->
+-----+-------------------+----------+
¦     ¦       .....       ¦   .....  ¦
+-----+-------------------+----------+
¦     ¦    Attribute N    ¦      ----+------->
+-----+-------------------+----------+
```

Figure 6.1 A Non-Directory-Attribute Table (NAT)

```
+-----------------------+----------+
¦    Attribute-level    ¦ Pointer  ¦
¦  access controls for  ¦ to next  ¦
¦      Attribute-A      ¦  block   ¦
+-----------------------+----------+
¦<10,20,-,Job,No_Update>¦          ¦
+-----------------------+----------+
¦ <10,50,-,-,No_Delete> ¦          ¦
+-----------------------+----------+
¦     ......            ¦  .....   ¦
+-----------------------+----------+
¦ <40,50,-,-,No_Delete> ¦       ---+---> NULL
+-----------------------+----------+
```

Figure 6.2 An Attribute-Security-Specification Table (ASST) on Attribute-A for User i.

```
         NAT for User i
+-----+-------------------+-------+
|Next |   Non-Directory   |Pointer|
|Attr |     Attribute     |to ASST|
+-----+-------------------+-------+
|     |    Attribute A    |   ----+---------+
+-----+-------------------+-------+         |
|     |    Attribute B    |   ----+-->      |
+-----+-------------------+-------+         |
|     |       .....       | ..... |         |
+-----+-------------------+-------+         |
|     |    Attribute N    |   ----+-->      |
+-----+-------------------+-------+         |
                                            |
                                   ASST for |Attribute A
                                            V
                     +-----------------------+---------+
                     |    Attribute-level    | Pointer |
                     | access controls for   | to next |
                     |      Attribute-A      |  block  |
                     +-----------------------+---------+
                     |<10,20,-,Job,No_Update>|         |
                     +-----------------------+---------+
                     | <10,50,-,-,No_Delete> |         |
                     +-----------------------+---------+
                     |         ......        |  .....  |
                     +-----------------------+---------+
                     | <40,50,-,-,No_Delete> |     |   |
                     +-----------------------+-----+---+
                                                   |
                                                   V
                                                  NULL
```

Figure 6.3 NAT and Corresponding ASST for User i

values are examined by the record processing function. In the physical data operation part of record processing, the records are fetched from the secondary storage a track at a time. Each record in the track is examined to see that it satisfies the query in the request. For satisfying records, the operations necessary for the request type (insert,delete,update, retrieve) are performed. Since the postprocessing deals with the field-level values of the records, we can perform postprocessing in the physical data operation part of the record processing function. Consequently, we will perform the postprocessing operation for the records satisfying the query.

Before record processing is started, we will look at the user-index. If the pointer to the NAT is null in the user-index, then record processing will not have any postprocessing checks. If the user does have a pointer to the NAT, then we search the NAT. The NAT stores the non-directory attributes which have security specifications on them and a pointer to the ASST for the corresponding attributes. We fetch all the ASSTs with the pointers provided from NAT. All the attribute security specifications in the ASSTs are brought into the backend's main memory, since we will use all this access control information for each record. Now, we are ready to perform the record processing with postprocessing capability. For each record, if it satisfies the query in the request, then we will perform the postprocessing checks on it. This operation will be described in the following section.

E. THE POSTPROCESSING AUTHORIZATION PROCESS

The authorization process in postprocessing is the same as the authorization process in preprocessing (Chapter IV). The only difference is: we deal with records in postprocesing whereas we deal with clusters in preprocessing. This process is done as follows. For each attribute security specification obtained from ASSTs, the record's corresponding non-directory attribute-value is fetched. If the attribute value is not between the upper and the lower values of the attribute security specification, then we conclude that the record can be authorized for this attribute security specification. The same operation is performed for the next attribute security specification. If the attribute-value is between the upper and the lower values of the attribute security specification, then we perform exactly the same authorization process of preprocessing for each request type (see Section IV-A-2).

VII. CONCLUSION

In this thesis, we have described the design and analysis of an access control mechanism for the multi-backend database system (MDBS). The design of this access control mechanism is based on two major objectives. The first objective is to introduce an access control capability to MDBS without changing the existing parts of the system. The second objective is to have an access control mechanism without adding excessive overhead to the system. To accomplish these objectives, the access control mechanism is incorparated into the directory management. Thus, access control is performed before any record is retrieved.

In addition, the capability to protect information which does not have directory has been provided. This capabilities can be exercised by the database creator. The user is allowed to create a new Type C descriptor with an insert request, if the user is authorized to create a new descriptor on the corresponding attribute. The access control information for the new descriptor is specified by the database creator. Furthermore, the database creator is provided with the capability to modify the access control information.

As an access control principle, if the request is partially authorized, then the partial result is given to the user with a warning. For example, let us assume that the user wants to retrieve the names of those employees whose salaries are higher than $2000. However, the user is only authorized to learn the names of those whose salaries are up to $5000. In this case, MDBS will give the user the names of those whose salaries are between $2000 and $5000 and will warn the user that there are other names which are not authorized for the user.

The access control information for each user is stored in the descriptor-to-descriptor-based-security-specification table (DSST), the cluster-to-cluster-based-security-specification table (CSST) and the non-directory-attribute table (NAT). Entries in these tables are reached by following pointers in the user-index. For simplicity, we stored the user-index in the backends' main memory. For a database with a large number of users, it might not be feasable to store the user-index in the backends' main memories. The user-index can then be stored in the backends' secondary storages and the user's portion can be brought into the main memories at the log-on time.

Finally, the access control mechanism of MDBS has been designed to handle the value-dependent and pattern sensitive information (see Chapter I). It also provides protection for statistical data. Furthermore, it overcomes the pass-through problem, since records in a cluster are protected in the same way (i.e., only the authorized records are accessed. The unauthorized records are never passed-through the access control mechanism, nor are they accessed. See Chapter III).

LIST OF REFERENCES

1. Hsiao, D. K., Kerr, D. S., Madnick, S. E., Computer Security, ACM Monograph Series, 1979.

2. Menon, M. J., Hsiao, D. K., "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I)", Technical Report OSU-CISRC-TR-81-71, The Ohio State University, 1981

3. Menon, M. J., Hsiao, D. K., "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part II)", Technical Report OSU-CISRC-TR-81-8, Naval Postgraduate School, 1983.

4. Kerr, D. S., Orooji, A., Zong-Zhi, S., "The implementation of a Multi-Backend Database System (MDBS): Part II - Software Engineering strategies and efforts towards a prototype MDBS.", Technical Report OSU-CISRC-TR-82-1, Naval Postgraduate School, 1983.

5. He, X., and others, "The Implementation of a Multi-Backend Database System (MDBS): Part II - The Design of a Prototype MDBS", Advanced Database Machine Archtectures, Prentice-Hall, 1983, pp. 327-385

6. Boyne, R. D., and others, "The Implementation of a Multi-Backend Database System (MDBS): Part III - The Message-Oriented Version with Concurrency Control and Secondary-Memory-Based Directory Management", Technical Report N00014-75-C-0573, Chief of Naval Research,1983.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defence Technical Information Center 2
   Cameron Station
   Alexandria, Virginia 22314

2. Library, Code 0142 2
   Naval Postgraduate School
   Monterey, California 93943

3. Department Chairman, Code 52 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93943

4. Curricula Officier, Code 37 1
   Computer Technology
   Naval Postgraduate School
   Monterey, California 93943

5. Professor David K. Hsiao, Code 52 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943

6. Dr. Douglas Kerr, Code 52 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943

7. Turk Deniz Kuvvetleri 3
   Egitim Daire Baskanligi
   Bakanliklar Ankara TURKEY

8. Ltjg. Ali EKICI 2
   Dolapdere
   Kinalikeklik Sok No=31
   Kurtulus- Istanbul
   TURKEY

9. Bogazici Universitesi 1
   Bilgisayar Muhendisligi
   Bebek- Istanbul
   TURKEY

5. 227

208345

Thesis
E2783 Ekici
c.1 Detail design and analysis of an access control system for a multi-backend database system.